

Fault-Tolerant Storage and Quorum Systems for Dynamic Environments

M.Sc. Thesis

Uri Nadav

Uri.Nadav@weizmann.ac.il

Advisor: Moni Naor

Weizmann Institute of Science
Rehovot 76100, Israel

January 6, 2005

Abstract

We deal with storage systems and quorum systems for a dynamic environment where servers may join and leave the system. We suggest a file storage system construction based on the ‘*And-Or*’ quorum system, that has a $O(\sqrt{n})$ write complexity, $O(\sqrt{n} \log n)$ read complexity and a *constant* data blowup-ratio, where n represents the number of processors in the network. Our construction is fault-tolerant against an adversary that can crash $\theta(n)$ processors of her choice while having slightly less *adaptive* queries than the reader.

When both the legitimate reader and the adversary are nonadaptive we derive lower bounds on the read complexity, write complexity and data blowup ratio. We show these bounds are tight using a simple storage system construction, based on an ϵ -intersecting quorum system.

In the random-fault model we show that the And-Or quorum system possesses optimal algorithmic probe complexity for both non-adaptive and adaptive readers. The non-adaptive algorithm for finding a live quorum achieves a $O(\sqrt{n} \log n)$ algorithmic probe complexity which matches a lower bound of Naor and Wieder [20]. The adaptive algorithm finds a live quorum with a probe complexity which is linear in the size of a quorum ($O(\sqrt{n})$) and requires at most $O(\log \log n)$ rounds. To the best of our knowledge, this is the smallest number of rounds in which a live quorum can be found, using only $O(\sqrt{n})$ probes, for a system with an optimal load.

Last we present an adaptation of the above storage system and a quorum system for a dynamic environment. Both are based on the And-Or tree and a dynamic overlay network that emulates the De-Bruijn network. These adaptations maintain the good properties of the above constructions (e.g., fault-tolerance, load and availability). The algorithms suggested for the maintenance of these dynamic data structures are strongly coupled with the routing scheme of the network. This fact enables the use of gossip protocols which saves in message complexity and keeps the protocols simple and local.

Acknowledgements

I would like to thank my advisor Moni Naor for many useful conversations and for inspiring me with ideas. I am also very grateful for Moni's help in turning my research results into a formal paper. I would like to thank Gera Weiss for many useful discussions and comments. Last but not least, I would like to thank Ilan Gronau for his extensive comments on a draft of this thesis, which led to significant improvements in the presentation and for many useful discussions. Finally I would like to thank all the colleagues and friends that made my time at the Weizmann Institute a great experience.

Contents

1	Introduction	5
1.1	Motivation	5
1.2	Definitions and Preliminaries	6
1.3	A Simple Example of a Fault-Tolerant Storage System	8
1.4	Related Work	8
1.5	Results and Thesis Organization	10
1.5.1	Organization	11
2	Storage Systems with Non-Adaptive Retrieval	13
2.1	The Non-Adaptive Retrieval Model	13
2.2	Read-Write Tradeoff of a Storage System	17
2.3	Bounding the Blowup-Ratio	20
3	Fault-Tolerant Adaptive Storage System	22
3.1	Small Load Implies Some Resiliency Against a Non-adaptive Adversary	23
3.2	A Generic Scheme for a Fault-Tolerant Storage System With a Constant Blowup Ratio	24
3.3	Adaptive Storage System Based on the ‘ <i>And-Or</i> ’ System	25
3.3.1	Adaptive Read Strategy for Finding a Complete And Set.	25
3.3.2	Adaptive Read Strategy in the Presence of Faults.	28
3.3.3	Dynamically Adjusting the Retrieval Scheme to the Number of Faults.	30
4	Dynamic Implementation of the And-Or Storage System	32
4.1	Some Properties of the Distance Halving Dynamic Network	34
4.2	Dynamic fault-tolerant Scheme	35
4.2.1	Network Implementation of Storage and Retrieval	36

5	The And-Or Quorum System In a Random Faults Model	38
5.1	Motivation and Definitions	38
5.2	Algorithmic Probe Complexity of the And-Or Quorum System.	39
5.2.1	A Non-adaptive Algorithm.	40
5.2.2	An Adaptive Algorithm	42
5.3	The And-Or Quorum System on a Balanced Binary Tree	45
6	The Dynamic And-Or Quorum System	49
6.1	A dynamic And-Or Quorum System Using the Identity-Management Scheme	49
6.2	A Dynamic And-Or Quorum System Over the Distance Halving Network . .	52
6.2.1	The Dynamic And-Or Quorum System Based on a Smooth Distance Halving Network	52
6.2.2	The Adaptation of a Quorum to Network Updates.	53
6.2.3	Analyzing the Availability and Probe Complexity of the Dynamic And-Or.	54
6.3	Comparing with Other Dynamic Quorum Constructions.	58
7	Conclusions and Open Questions	59
7.1	Storage Systems in Adversarial Fault Model	59
7.2	Algorithmic Probe Complexity of Quorum Systems	60

List of Figures

3.1	And-Or tree after being pruned	29
3.2	ANDset' for early stopping	31
4.1	Edges of the Distance-Halving graph	35
4.2	Balanced Network	36
5.1	Internal ANDset	41
5.2	Quorum in A Balanced Tree	46
6.1	Processors Join and Depart from the tree	51

Chapter 1

Introduction

1.1 Motivation

We deal with methods for constructing distributed storage systems over peer-to-peer networks, i.e. loosely coupled environments where servers join and leave the system dynamically. Such networks are subject to various types of faults and malicious attacks and hence providing even simple functionalities is a challenge. Such a ‘*storage system*’ should provide a distributed file system on a set of processors (computers with storage capabilities), intended to store multiple files and to be accessed by many users. The two natural components of such a system are the *storage* mechanism that indicates how to distribute files on processors and the *retrieval* mechanism that allows file retrieval.

The storage process is composed of two steps. First, selecting processors that store data, and second, encoding a file into bits, which are then written to the selected processors. The retrieval consists of reading bits from processors and decoding them, so as to obtain the stored files. The procedures for selecting the processors to which data is written and from which data is read are referred to as ‘*write strategy*’ and ‘*read strategy*’ respectively. We distinguish between non-adaptive and adaptive strategies. In a non-adaptive strategy the set of processors is determined prior to any access, whereas in an adaptive strategy, some queries are allowed before the set is determined. A storage system is considered resilient to an adversarial attack, if each file stored can be retrieved after the attack.

Our work considers the fault-tolerance of storage systems and quorum systems in both the random-fault model and in an adversarial model. In a random-fault model each processor independently fails with some predefined constant probability. In an adversarial model the failing processors are chosen maliciously. The adversary can be characterized by the amount of control it has over choosing the faulty processors and the amount of control it has over their

behavior (fail-stop, Byzantine). Note that the ability to choose the faulty processors does not imply total control over their behavior. The type of faults considered here are fail-stop (or data deletion). A major issue in the adversarial-fault model is the degree of adaptiveness the adversary has in selecting which processors to delete and query. One may consider a whole spectrum of adaptive behaviors. Most of our results consider an adversary who is allowed to delete the storage of up to t processors of her choice, selected in a non-adaptive manner.

We define a storage system as fault-tolerant if each file can be reconstructed with high probability (w.h.p.) after faults have been caused. This model characterizes better tolerance against a *sensor*, because a sensor may wish to target a small number of files and not only eliminate access to files on average.

We first consider storage systems in a static network model, for which the set of processors is fixed. We then proceed to discuss dynamic networks, where a major problem is finding processors that currently are in the network. Several designs [13, 19, 26, 29, 31] deal with this problem.

1.2 Definitions and Preliminaries

A *quorum system* (QS) \mathcal{Q} over a universe U of processors, is a collection of subsets of U , every two of which intersect. The subsets in the collection are often referred to as quorums. An *access strategy* μ , for a quorum system \mathcal{Q} , is a probability distribution over the quorums in \mathcal{Q} , i.e. $\sum_{Q \in \mathcal{Q}} \mu(Q) = 1$. An access strategy can be thought of as a method for choosing a quorum. An ϵ -*intersecting quorum system*, as defined by Malkhi et al. [16], is a tuple (\mathcal{Q}, μ) of a set system and an access strategy μ , such that $\Pr[Q \cap Q' \neq \emptyset] > 1 - \epsilon$, where the probability is taken with respect to μ .

The *load* on a system of processors, as defined in [22], captures the probability of accessing the busiest processor. The load $\mathcal{L}(i)$ induced by a strategy on a processor $i \in U$ is the probability that processor i is chosen by the strategy, i.e. $\mathcal{L}(i) = \Pr_{\mu}[i \in Q]$. The load of a strategy μ is $\mathcal{L}(\mu) = \max_{i \in U} \mathcal{L}(i)$. For a quorum system \mathcal{S} , the load $\mathcal{L}(\mathcal{S})$ is the minimum load induced by a strategy, over all strategies. Let c denote the size of the smallest quorum. Naor and Wool prove in [22] the following lemma:

Lemma 1.1 (from [22]). *The load of a quorum system \mathcal{S} is at least $\max\{\frac{1}{c}, \frac{c}{n}\}$ which implies that $\mathcal{L}(\mathcal{S}) \geq \frac{1}{\sqrt{n}}$.*

From here, whenever we refer to a system with an optimal load, we mean a load of $O(\frac{1}{\sqrt{n}})$. It was shown in [16] that this lower bound on the load also holds for an ϵ -intersecting quorum system.

A quorum is said to be alive if all of its elements are alive, and dead otherwise. The *availability* of a quorum system \mathcal{Q} is measured by the failure probability F_p which is the probability that all quorums in \mathcal{Q} are dead, when each processor fails independently with probability p . This failure probability measures how resilient the system is. In [23] Peleg and Wool study the availability of some known quorum systems. In [22] it was shown that both the Paths quorum system and the And-Or quorum system have a failure probability which exponentially decays in n , for small enough processor failure probability.

Quorum systems are often used in distributed computation. A distributed storage system can be trivially constructed using any quorum system \mathcal{Q} with an access strategy μ : To store a file, a quorum is selected in accordance with μ and a copy of the file is written to each of the processors in the chosen quorum. The retrieval scheme consists of selecting a quorum in accordance with μ and querying each of the processors in the chosen quorum for the file which is being retrieved. By the intersection property of quorum systems, the reconstruction of a file is then guaranteed. Notice that the same construction holds for any ϵ -intersecting quorum system (\mathcal{Q}, μ) , in which case the reconstruction is guaranteed with probability at least $1 - \epsilon$. The reverse claim is also true in the sense that every storage system can serve as a quorum system, since a set of processors which is used for writing a file must intersect every set of processors that is used for reading that file. In Chapter 2 we show that no storage system can do better than this simple construction, in a non-adaptive model. In Chapter 3 we present a storage system that outperforms this quorum based solution, in terms of data blowup ratio, but in a slightly modified model. This is the reason we define and investigate storage systems rather than just deal with quorum systems.

Storage systems are measured by the following parameters:

Write/Read Complexity: The average number of processors accessed by the write/read strategies of the storage system, in a write/read operation of a single file.

Blowup-Ratio: The ratio between the total number of bits used in the storage of a file, and its size.

Resiliency: The strongest fault model in which the system is resilient.

A design goal of storage systems is to minimize the write/read complexity and the blowup-ratio. However, as we shall see, these goals are somewhat contradictory when the resiliency of the system is taken into considerations.

1.3 A Simple Example of a Fault-Tolerant Storage System

Malkhi et al. give in [16] an explicit construction for an ϵ -intersecting quorum system. The quorums are all the subsets of U of size $\ell\sqrt{n}$, where ℓ depends only on ϵ . The access strategy is the uniform distribution over all quorums. The storage system constructed using this quorum system has a $\ell\sqrt{n}$ blowup-ratio and write/read complexity.

Suppose a non-adaptive adversary can delete a set of processors T of size δn of her choice, where $0 < \delta < 1$ is a constant. Because of the independence between the faulty set T , and the quorums chosen for storage and retrieval, w.h.p. no more than δ fraction of a quorum is deleted. It is then possible to select ℓ in accordance with δ so as to maintain the ϵ -intersection property in the presence of such faults. In [2] Abraham et al. show how to adapt a quorum in this construction to a dynamic environment as processors join and leave, guaranteeing an ϵ -intersection property.

Another virtue of this storage system is that it is resilient even against an adaptive adversary. An adversary that reads the memory of some processors and knows what files they store, gains no information on the distribution of a file among the remaining processors.

1.4 Related Work

Fault Tolerant Data Networks

There are many models for fault-tolerant data networks. In a worst case model, the existence of an adversary is assumed. The adversary knows the memory content of each processor and may corrupt a constant fraction of the processors. In this case the storage system can be thought of as an error correcting code, where the message is a concatenation of all the files stored in the system. Therefore, it follows that the write complexity of such a scheme, which is fault-tolerant to $\Omega(n)$ faults, must be $\Omega(n)$. This is a direct consequence of the fact that the distance between two codewords in an error correcting code, resilient against a linear number of faults must be linear in the number of symbols.

Rabin defines an Information Dispersal Algorithm (IDA) in [25] as a storage scheme of a file over n processors, such that any m of them can recover the file. The storage operation accesses all n processors. During retrieval, actual data is read from m of the n processors, but still, a reader does not generally know in advance which processors survived, so it must try to access all n of them. The data blowup-ratio of Rabin's scheme is constant when m

is linear in n . Another way of viewing Rabin’s IDA, in our context, is to consider all the files stored in the system as the dispersed information. This however, will not reduce the read complexity of a file, since a single symbol from a dispersed message cannot generally be retrieved from fewer than m processors.

The linear write complexity in a worst case model is necessary, however, the read complexity need not be linear. There exist error correcting codes that enable the reconstruction of a single symbol of a message by looking at a limited number of bits of a (possibly) corrupted encoding. This property is often referred to as ‘local decodability’. Several lower bounds are available on the rate of a code that is locally decodable from a constant number of queries: Kerenidis and Wolf [10] give an exponential lower bound on the rate of 2 queries locally decodable code (codes in which each symbol in a message is reconstructible from reading two symbols of the message). Katz and Trevisan [9] show that a code that is locally decodable from a constant number of queries cannot have constant rate.

Reed-Muller codes, based on multivariate polynomials (c.f.[30]), have been known to have local decoding properties. There exist parameters of Reed-Muller codes that achieve constant rate such that the local decoding of a single symbol can be done with $O(\sqrt[h]{n})$ queries, where n is the length of the code, and h is some constant. Local decoding succeeds w.h.p. even with constant error rate (a rate proportional to h^h is achieved when considering multivariate polynomials with h variables and degree $\sqrt[h]{n}$).

A linear write and read complexity is not considered practical in a peer-to-peer network, where even the set of addresses of the actual participants is not assumed to be within reach. We will usually prefer higher data blowup-ratio, to prevent high write and read complexities.

A content addressable network (CAN) [26] or a distributed hash table (DHT), as described by Ratnasamy et al., is a distributed infrastructure that provides hash table like functionality on Internet like scales. Other constructions of DHT’s are given in [29, 31, 19, 13]. The works of [21, 29, 31] yield fault-tolerant DHT’s for a dynamic environment, in a random fault model, where the faults are uniformly distributed over the participating processors. Fault-tolerance in this case is achieved through replication of each data item to a logarithmic number of processors.

Saia et al. propose in [28] a content addressable network in which after an adversarial removal of a constant fraction of the processors only a small fraction of the files are not accessible. However, a censor who wishes to eliminate access to a specific file can do so by crashing $\log n$ processors. In contrast, we are concerned with a model in which every file can be recovered w.h.p. after adversarial deletions.

Alternatively to the above examples, we consider a fault-model that is stronger than

the random-fault model, but weaker than the worst case model. In Chapters 2, 3 and 4 we assume the existence of an adversary who tries to eliminate access to a stored file, by crashing a set of processors of her choice. The general assumption, however, is that the adversary has only limited knowledge of the content of processors' memories. We feel that it is reasonable to assume this type of adversarial behavior in Internet scale networks.

Quorum Systems

In Chapters 5 and 6 we consider a random-fault model and study the fault-tolerance of quorum systems in classic and dynamic environments. Peleg and Wool defined the availability of quorum systems in [23] and studied the availability of some quorum system classes. In [22] Naor and Wool presented the *Paths* quorum system, and showed that it is a highly available quorum system. In [20], Naor and Wieder suggested an adaptation of *Paths* in a dynamic environment and showed that it maintains the optimal load and availability of the original system.

In [24] Peleg and Wool studied the complexity of finding a live quorum or evidence for the lack of, in an adversarial-fault model. In this model crash faults are assumed to be detectable, and a probe action to a processor is assumed to have $O(1)$ complexity. In [6] Hassin and Peleg analyzed the probe complexity of several systems in the random-fault model. Bazzi [4] introduced the term '*cost of failures*'. Given a network implementation and an algorithm for finding a live quorum, the cost of failures measures the average communication overhead caused by encountering a faulty processor. In [20] Naor and Wieder defined the *Algorithmic probe complexity* as the actual time and message complexity required for finding a live quorum.

When dealing with quorum systems in a dynamic environment, we use the algorithmic probe complexity as a primary measure of quality, mainly because it takes into consideration the network implementation, which is of major interest in dynamic networks. Finally, we will compare the algorithmic probe complexity of our construction to that of existing ones.

1.5 Results and Thesis Organization

Our work considers storage systems and quorum system in different fault models. For the adversarial fault-model our results split under two categories:

Good news: When the read strategy is slightly more powerful than the adversary (for example by using adaptive queries), a construction of a fault-tolerant storage system

with constant blowup-ratio, write complexity $O(\sqrt{n})$ and read complexity $O(\sqrt{n} \log n)$ is given. We also present a storage system where the read complexity is dynamically adjusted to the number of faults so as to allow early stopping of the read process in the case less faults occur. These constructions are implemented in a dynamic environment (peer-to-peer like), yielding a fault-tolerant storage system with constant blowup-ratio. Our construction for the dynamic environment is strongly coupled with the routing mechanism of the ‘Distance Halving’ network [19]. This allows the storage algorithms to be implemented through a ‘gossip protocol’ on the edges of the network. This approach both saves in latency and message complexity, and keeps the construction simple and easy to implement.

Bad news: No fault-tolerant storage system with non-adaptive read strategy can do better than the construction discussed in Section 1.3. In particular, it will be shown in Chapter 2 that there exists a tradeoff between write complexity and read complexity, and that the above construction is optimal in this case. It is further shown that there exists a tradeoff between the blowup-ratio and the read complexity. Specifically, non-adaptive storage systems that are fault-tolerant against t non-adaptive deletions and have a blowup-ratio ρ , must have read complexity $\Omega(\frac{t}{\rho})$.

A second set of contributions is for the random-fault model. We analyze the algorithmic probe complexity of the ‘And-Or’ quorum system suggested by Naor and Wool [22], in a classical environment (where there are exactly n processors throughout the lifetime of the network). We show that the And-Or system has optimal algorithms for finding a live quorum, both when the reader may or may not act in an adaptive way. Our non-adaptive algorithm has an algorithmic probe complexity of $O(\sqrt{n} \log n)$ which matches a lower bound of Naor and Wieder [20]. Our adaptive algorithm finds a live quorum with a probe complexity which is linear in the size of a quorum ($O(\sqrt{n})$) and requires at most $O(\log \log n)$ rounds. To the best of our knowledge, this is the smallest number of rounds in which a live quorum can be found, using only $O(\sqrt{n})$ probes, for a system with an optimal load.

1.5.1 Organization

The remainder of this thesis is organized in the following way. Chapter 2 discusses fault-tolerant storage systems for non-adaptive retrieval model. We show that in such systems there exists a tradeoff between write complexity and read complexity, and between the blowup-ratio and read complexity. More specifically, we show a lower bound to the product of the read complexity and write complexity as a function of the size of the faulty set (and

a lower bound to the product of the blowup-ratio and read complexity). The example of Section 1.3 shows that these tradeoffs are tight. In Chapter 3 we present a fault-tolerant storage system, with an adaptive read strategy that maintains a constant blowup-ratio. Both these chapters deal with the storage of a single file in a static network. In Chapter 4 we discuss an adaptation of the above scheme to a dynamic model that yields a fault-tolerant storage system, for the storage of multiple files, in a dynamic environment. A gossip protocol is offered for the implementation, keeping our construction simple and easy to implement in a dynamic environment. Chapters 5 and 6 consider the And-Or quorum system in the random-fault model. In Chapter 5 we present the optimal adaptive and non-adaptive algorithms for finding a live quorum in the random-fault model. In Chapter 6 we present two different implementations of the And-Or system in a dynamic environment.

Chapter 2

Storage Systems with Non-Adaptive Retrieval

This chapter deals with the analysis of storage systems with non-adaptive retrieval schemes. A non-adaptive read strategy determines which processors to probe prior to accessing any processor. A tradeoff between write complexity and read complexity is shown, and a lower bound on the blowup ratio is given.

2.1 The Non-Adaptive Retrieval Model

We begin by giving formal definitions of a storage system with non-adaptive retrieval, and of fault-tolerance in the non-adaptive adversary model. The storage of a file is done in accordance with some *write strategy*, which defines the number of bits that are stored in each processor. Similarly, the retrieval of a file is done in accordance with some *read strategy*, which defines the set of processors from which data is read in the retrieval process.

Definition 2.1 (Write/Read Strategy). A write strategy μ_w for a universe U of size n is a probability distribution on \mathbb{N}^n . A read strategy μ_r for U is a probability distribution on $\{0, 1\}^n$.

Let q_w, q_r be the vectors generated by the write and read strategies respectively. q_w represents the number of bits allocated in each processor for the storage of a file. The support set of q_r (q_w) represents the set of processors which are being accessed during retrieval (storage). The inner product of q_w and q_r , denoted by $\langle q_r, q_w \rangle$ represents the total number of bits read during the retrieval process (or more precisely, the inner product represents the total number of bits read, from the set of actual bits that were explicitly stored). The

support set of a vector is the set of processors involved in a write/read operation represented by it. We sometimes abuse notation by alternating between a vector and its support set, and denote by $|q|$ the size of the support set. We define the *write/read complexity* of a system as the expected number of processors accessed during a write/read operation.

Similar to the notion of load as defined for quorum systems, the ‘*write load*’, \mathcal{L}_w , induced by a write strategy μ_w is the maximum probability of choosing a processor by the strategy:

$$\mathcal{L}_w \triangleq \max_{1 \leq i \leq n} \{\Pr[q_w(i) \neq 0]\}.$$

The ‘*read load*’ is defined similarly and is denoted \mathcal{L}_r .

The storage of a file includes memory allocation in a set of selected processors, and encoding of the file into the allocated storage space. The retrieval of a file f includes reading encoded data from the memories of a selected set of processors and decoding this information to retrieve f . A processor that holds no data about f can also be a member of the probed set. In this case it returns *null*.

Definition 2.2 (Non-Adaptive Elementary Storage System). *A k -elementary storage system is a 4-tuple $(\mu_w, \mu_r, \mathcal{E}, \mathcal{D})$ consisting of*

Access Strategies: *A write strategy μ_w and a read strategy μ_r .*

Encoding mapping:

$$\mathcal{E}(f, q_w) \mapsto (x_1, \dots, x_n),$$

where $q_w \in \mathbb{N}^n$ and

$$x_i \in \begin{cases} \{0, 1\}^{q_w(i)}, & q_w(i) > 0, \\ \{*\}, & \text{otherwise.} \end{cases}$$

This mapping defines the encoding rule from a file of size k bits and a list of memory allocations into processors’ memories. A ‘’ represents that no data is stored in a processor.*

Decoding (reconstruction) mapping:

$$\mathcal{D}(x_1, \dots, x_n) \mapsto \{0, 1\}^k.$$

*where x_i is either a bit string or one of the symbols $\{\phi, *\}$. A ‘*’ in the i th parameter represents that no data was written to the i th processor. A ‘ ϕ ’ in the i th parameter represents that the i th processor was not queried.*

Notice that definition 2.2 does not deal with the correctness of reconstruction. Definition 2.3 below, characterizes non-adaptive storage systems in which a file is successfully reconstructed i.e., the decoding of a file from it's encoded data equals the file itself.

Let π be the projection from a list of n parameters (x_1, \dots, x_n) and a characteristic vector $q \in \{0, 1\}^n$ onto a list of n items such that the i th item is:

$$(\pi((x_1, \dots, x_n), q))_i = \begin{cases} x_i, & q(i) = 1, \\ \phi, & \text{otherwise.} \end{cases}$$

The projection of (x_1, \dots, x_n) by q models that only processors which belong to the support set of q were queried.

The result of the process of encoding a file f using a list of memory allocations q_w and then reading the memories of processors using q_r and decoding, is described as $\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r))$. If $\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r)) \equiv f$, then we say f was recovered successfully.

Definition 2.3 (Non-Adaptive (ϵ, k) -Reconstructible Storage System). *An (ϵ, k) -reconstructible storage system is a storage system $\mathcal{S} = (\mu_w, \mu_r, \mathcal{E}, \mathcal{D})$ such that:*

$$\forall f \in \{0, 1\}^k, \Pr[\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r)) \equiv f] > 1 - \epsilon,$$

where $q_w \sim \mu_w, q_r \sim \mu_r$ are independent random variables.

We refer to Non-Adaptive (ϵ, k) -Reconstructible Storage System simply as (ϵ, k) -Storage System.

Notice that information can be encoded implicitly in the way bits are allocated among processors and not only in the stored bits themselves. For example, consider the following storage system: A file is encoded using a total of ℓ bits, allocated between all n processors. The retrieval includes reading from each processors. Since there are $\binom{\ell+n-1}{n}$ different allocations of ℓ bits to n processors, a file of size $\ell + \log \binom{\ell+n-1}{n}$ can be stored and retrieved in such a system. The number of saved bits encoded this way can have only a relatively small contribution, when the number of processors used for writing/reading is sufficiently small relative to the number of stored bits. We therefore choose to ignore storage systems that make use of such implicit encodings, and consider storage systems in which the decoding procedure of the retrieval process uses only the bits that were explicitly stored. These are systems in which the distribution μ_w from which the memory allocation vector q_w is sampled, is fixed and does not change when different files are stored.

Remark 2.4. *Notice that definition 2.2 of an elementary storage system, with a fixed write strategy μ_w , already captures the case that the memory allocations vector is independent of the content the stored file.*

Theorem 2.6 states a necessary information theoretic condition on the read and write strategies of an (ϵ, k) storage system. Such a system must w.h.p. read at least k bits of information during retrieval. Definition 2.5 captures this property of a pair of read and write strategies.

Definition 2.5 ((ϵ, k)-Intersection Property). Let (μ_w, μ_r) be a pair of a write strategy and a read strategy on a set of processors. We say that μ_r, μ_w satisfy the (ϵ, k) -intersection property, if

$$\Pr[\langle q_w, q_r \rangle \geq k] \geq 1 - \epsilon,$$

where $q_w \sim \mu_w$ and $q_r \sim \mu_r$ are independent random variables.

Theorem 2.6. Let $\mathcal{S} = (\mu_w, \mu_r, \mathcal{E}, \mathcal{D})$ be an (ϵ, k) storage system. Then the write and read strategies (μ_w, μ_r) must satisfy the $(2\epsilon, k)$ intersection property.

Proof. We show the necessity of the $(2\epsilon, k)$ intersection property by showing the converse. Namely, if the $(2\epsilon, k)$ intersection property does not hold, i.e. $\Pr[\langle q_w, q_r \rangle < k] > 2\epsilon$, then there exist a file f' for which

$$\Pr[\mathcal{D}(\pi(\mathcal{E}(f', q_w), q_r), q_r) \neq f'] > \epsilon.$$

where the probability is taken over the choices of write and read vectors (q_w, q_r) .

To this end, we state the following fact. For a fixed pair (q_w, q_r) , since the number of different inputs for \mathcal{D} is at most $2^{\langle q_r, q_w \rangle}$, it follows that if $\langle q_w, q_r \rangle < k$, then \mathcal{D} must err on at least 2^{k-1} files of the 2^k possible files. This is easily seen by observing that $2^{\langle q_w, q_r \rangle}$ different inputs can produce at most $2^{\langle q_w, q_r \rangle}$ different outputs. Therefore, if one assumes that all files are equally likely to be stored, then

$$\Pr[\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r), q_r) \neq f \mid \langle q_w, q_r \rangle < k] \geq \frac{1}{2}$$

(where the probability is taken over all choices of files and write and read vectors), and consequently

$$\begin{aligned} \Pr[\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r), q_r) \neq f] &= \Pr[\langle q_w, q_r \rangle \geq k] \Pr[\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r), q_r) \neq f \mid \langle q_w, q_r \rangle \geq k] \\ &\quad + \Pr[\langle q_w, q_r \rangle < k] \Pr[\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r), q_r) \neq f \mid \langle q_w, q_r \rangle < k] \\ &> 2\epsilon \Pr[\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r), q_r) \neq f \mid \langle q_w, q_r \rangle < k] \\ &\geq \epsilon. \end{aligned}$$

Finally, since the probability of incorrect decoding of a randomly chosen file is greater than ϵ , it follows that there exists at least one file whose probability of incorrect decoding is greater than ϵ . This proves the converse and the theorem as a whole. \square

The $(2\epsilon, k)$ intersection property was shown to be a necessary condition for an (ϵ, k) storage system, however, it is not necessarily sufficient. In order to ensure recovery, the retrieval process must yield the correct k bits, i.e. the encoding and decoding schemes must be considered. It turns out that the necessary condition is ‘almost’ sufficient. Suppose each stored symbol represents the solution to a linear equation over the k bits of a file. To reconstruct the file, it is sufficient to collect the solutions to a set of equations of rank k . It is known that w.h.p., $(1 + \gamma)k$ random linear equations over k variables, have a rank k , where γ is a small constant. Hence, by simply storing a random linear combination of the file’s bits at each stored symbol, it is sufficient to collect $(1 + \gamma)k$ symbols. Naor and Roth [18] investigate the problem of file distribution in a network. The problem there is different, but the coding methods they suggest can also be used in this context.

For this reason we do not consider any specific coding scheme and simply investigate the pair (μ_w, μ_r) of write and read strategies for which the (ϵ, k) -intersection property holds.

The following definition addresses fault-tolerant storage systems in the non-adaptive adversary model.

Definition 2.7 (Non-adaptive Fault-Tolerant Storage System). *A k -storage system with write and read strategies μ_w, μ_r is said to be (ϵ, t) -fault-tolerant, if for any set of deleted processors T of size less than t , the induced system (the write and read strategies restricted to $U \setminus T$), satisfies the (ϵ, k) -intersection property. We denote such a system as a (t, ϵ, k) -storage system.*

Definition 2.7 matches an adversary that can choose (non-adaptively) the set of faulty processors, but crashes the processors in a fail-stop manner, i.e. she does not have full control over the program of processors. The reason for choosing this hybrid adversarial behavior is twofold: First, we are dealing now with lower bounds, for which it is sufficient to assume fail-stop faults. Second, such a model has a justification in the ‘real world’ of dynamic storage systems implementations, where it is relatively easy for a processor to gain a desired identity in the system. In those networks identities are usually given randomly, so by simply joining and leaving the network enough times, a processor can gain a desired identity with no need to hack/change the program.

2.2 Read-Write Tradeoff of a Storage System

Given a write strategy μ_w , we are interested in the best read strategy μ_r , in terms of read complexity, such that (μ_w, μ_r) satisfy the (ϵ, k) intersection property.

Lemma 2.8. *Let μ_w be a write strategy on a universe U of size n . For every read strategy μ_r such that (μ_w, μ_r) satisfy the (ϵ, k) -Intersection Property and $q_r \sim \mu_r$ we have*

$$\mathbb{E}[|q_r|] > \frac{1 - \epsilon}{\mathcal{L}(\mu_w)}.$$

Proof. Let $q_w \sim \mu_w$ and $q_r \sim \mu_r$ be independent random variables. By the intersection property, $\Pr[\langle q_w, q_r \rangle > k] > 1 - \epsilon$. Denote by Q_w and Q_r the sets induced by q_w and q_r , respectively. It is immediate that $\Pr[|Q_r \cap Q_w| \neq \emptyset] > 1 - \epsilon$. Hence,

$$\mathbb{E}[|Q_r \cap Q_w|] > \Pr[|Q_r \cap Q_w| \geq 1] > 1 - \epsilon. \quad (2.1)$$

Next, let x_i be an indicator random variable that equals 1 if element i belongs to the intersection $Q_r \cap Q_w$, and 0 otherwise. It follows that

$$\mathbb{E}[x_i] = \Pr[x_i = 1] = \Pr[i \in Q_r \cap Q_w] = \Pr[i \in Q_r] \cdot \Pr[i \in Q_w], \quad (2.2)$$

where the last equality is due to the independence of Q_r and Q_w . Finally, $\mathbb{E}[|Q_r \cap Q_w|]$ can be upper bounded in the following way.

$$\mathbb{E}[|Q_r \cap Q_w|] = \mathbb{E}\left[\sum_{i=1}^n x_i\right] = \sum_{i=1}^n \mathbb{E}[x_i] = \sum_{i=1}^n \Pr[x_i \in Q_r] \Pr[x_i \in Q_w] \leq \mathcal{L}_w \mathbb{E}[|Q_r|]. \quad (2.3)$$

where the third equality follows from (2.2) and the inequality follows from the definition of load. Combining (2.3) and (2.1) completes the proof of the lemma. \square

Next we turn to analyze fault-tolerant storage systems. We show that in such systems, where the retrieval scheme is non-adaptive, there exists a tradeoff between the read complexity and the write complexity. This is shown by using the power of the adversary to delete any set of size t , and specifically the set composed of the t most write loaded processors.

Theorem 2.9. *Let (μ_r, μ_w) be the write strategy and read strategy of a (t, ϵ, k) fault-tolerant storage system. Then*

$$\mathbb{E}[|q_r|] \cdot \mathbb{E}[|q_w|] \geq (1 - \epsilon)t,$$

where $q_r \sim \mu_r$ and $q_w \sim \mu_w$ are independent of each other.

Proof. Assume by contradiction that there exists a (t, k, ϵ) fault-tolerant storage system (μ_r, μ_w) such that:

$$\mathbb{E}_{\mu_r}[|q_r|] \cdot \mathbb{E}_{\mu_w}[|q_w|] < (1 - \epsilon)t.$$

Set $T' \triangleq \{u \in U \mid \mathcal{L}_w(u) \geq \frac{\mathbb{E}[\|q_w\|]}{t}\}$. The size of T' is less than t since by definition $\sum_{u \in U} \mathcal{L}_w(u) = \mathbb{E}[\|q_w\|]$, hence there can be at most t elements with load greater or equal than $\frac{\mathbb{E}[\|q_w\|]}{t}$.

By definition of (μ_w, μ_r) as a (t, k, ϵ) storage system, after the deletion of any set of indices T , and specifically T' , the system remains an (ϵ, k) storage system. Denote by (μ'_w, μ'_r) the derived system after the deletion of T' . Clearly $\mathbb{E}_{\mu_r}[\|q_r\|]$ is greater than or equal to $\mathbb{E}_{\mu'_r}[\|q_r\|]$, because deleted indices are set to 0. For the same reason $\mathbb{E}_{\mu_w}[\|q_w\|]$ is greater than or equal to $\mathbb{E}_{\mu'_w}[\|q_w\|]$. It follows from the construction of T' that the write load \mathcal{L}'_w , induced by μ'_w , satisfies

$$\mathcal{L}'_w < \frac{\mathbb{E}_{\mu_w}[\|q_w\|]}{t} \quad (2.4)$$

From Lemma 2.8, we have the following relation between the read complexity and the write load.

$$\mathbb{E}_{\mu'_r}[\|q_r\|] > \frac{1}{\mathcal{L}'_w} (1 - \epsilon) \quad (2.5)$$

Finally, combining (2.4) and (2.5), we get

$$\begin{aligned} \mathbb{E}_{\mu_r}[\|q_r\|] \cdot \mathbb{E}_{\mu_w}[\|q_w\|] &> \mathbb{E}_{\mu'_r}[\|q_r\|] \cdot \mathbb{E}_{\mu_w}[\|q_w\|] \\ &> \frac{1}{\mathcal{L}'_w} (1 - \epsilon) \mathbb{E}[\|q_w\|] \\ &> t(1 - \epsilon) \end{aligned}$$

contradicting the initial assumption. □

Sometimes it is desirable to require that the number of processors from which data is retrieved exceeds some threshold (τ). For example when the problem of Byzantine faults is considered and addressed using a majority vote [14, 15]. In that case, the intersection property is defined as $\Pr[|Q_r \cap Q_w| > \tau] > 1 - \epsilon$, where Q_r, Q_w are distributed as μ_r, μ_w respectively. A similar result to Theorem 2.9 holds in that case.

Theorem 2.10. *Let (μ_r, μ_w) be the write and read strategies of a (t, ϵ, k) fault-tolerant storage system with threshold τ . Then $\mathbb{E}[\|q_r\|] \cdot \mathbb{E}[\|q_w\|] \geq (1 - \epsilon)\tau t$, where $q_r \sim \mu_r$ and $q_w \sim \mu_w$ are independent of each other.*

Proof. Similar to proof of Theorem 2.9 □

2.3 Bounding the Blowup-Ratio

We bound the minimum number of bits used for the storage of a single file in the non-adaptive model. The following memory model for a processor is considered. It is assumed that each file is encoded and stored separately. In this model a file can be inserted or modified without reading any other file. This assumption (which is implicit in the non-adaptive model) simplifies the storage system and keeps it oblivious to past operations. In that sense, this model suits storage systems for dynamic and scalable networks.

The total storage used for a file is the number of bits assigned over all processors. The blowup-ratio of a storage system is the ratio between the average total storage, and the size of a file which is denoted by k .

Definition 2.11 (blowup-ratio). *The blowup-ratio $\rho(\mathcal{S})$ of an (ϵ, k) storage system \mathcal{S} with a write strategy μ_w , is defined as $\rho(\mathcal{S}) \triangleq \mathbb{E}_{\mu_w}[\|q_w\|]/k$, where $\|\cdot\|$ denotes the L_1 -norm.*

Obviously it is desirable to have as small as possible blowup-ratio. However, the next theorem shows there is a tradeoff between the read complexity and the blowup-ratio when the reader is non-adaptive.

Theorem 2.12. *Let \mathcal{S} be a (t, ϵ, k) storage system with a read strategy μ_r , then:*

$$\rho(\mathcal{S}) > (1 - \epsilon) \frac{t}{\mathbb{E}[\|q_r\|]},$$

where $q_r \sim \mu_r$.

Proof. Let μ_w be the write strategy of \mathcal{S} and let $q_w \sim \mu_w$ independent of q_r . By definition of a fault tolerant storage system $\Pr[\langle q_w, q_r \rangle > k] > 1 - \epsilon$. Thus,

$$\mathbb{E}[\langle q_w, q_r \rangle] \geq \Pr[\langle q_w, q_r \rangle > k] \cdot k > (1 - \epsilon)k. \quad (2.6)$$

Define the set T to be the set of the most read-loaded processors, i.e.,

$$T \triangleq \{u \mid \mathcal{L}_r(u) > \mathbb{E}[\|q_r\|]/t\}.$$

The sum of loads over all processors equals by definition $\mathbb{E}[\|q_r\|]$, thus T has at most t elements.

Set random variable q'_r defined as $[q'_r]_i = \begin{cases} [q_r]_i, & i \notin T, \\ 0, & \text{otherwise,} \end{cases}$

and define random variable q'_w in the same way. Clearly, by definition of T as the most read-loaded indices, $\Pr[[q'_r]_i = 1] \leq \frac{\mathbb{E}[\|q_r\|]}{t}$ for all $i \in U$. Hence, by definition, the load \mathcal{L}'_r

induced by q'_r is less than $\frac{\mathbb{E}[\|q_r\|]}{t}$. By definition of (μ_w, μ_r) as t fault-tolerant, q'_r, q'_w preserves the intersection property. hence,

$$\begin{aligned}
(1 - \epsilon)k &< \mathbb{E}[\langle q'_w, q'_r \rangle] \\
&= \mathbb{E}\left[\sum_{i=1}^n [q'_w]_i \cdot [q'_r]_i\right] \\
&\stackrel{(a)}{=} \sum_{i=1}^n \mathbb{E}[[q'_w]_i \cdot [q'_r]_i] \\
&\stackrel{(b)}{=} \sum_{i=1}^n \mathbb{E}[[q'_w]_i] \cdot \mathbb{E}[[q'_r]_i] \\
&= \sum_{i=1}^n \mathbb{E}[[q'_w]_i] \cdot \Pr[[q'_r]_i = 1] \\
&\leq \sum_{i=1}^n \mathbb{E}[[q'_w]_i] \cdot \mathcal{L}'_r \\
&\stackrel{(c)}{=} \mathcal{L}'_r \mathbb{E}[\|q_w\|] \leq \frac{\mathbb{E}[\|q_r\|]}{t} \mathbb{E}[\|q_w\|].
\end{aligned}$$

where (a) and (c) follows from the linearity of expectation and (b) follows from the independence of q'_r and q'_w . Thus, $\rho > \frac{(1-\epsilon)t}{\mathbb{E}[\|q_r\|]}$. \square

In particular, when $t = \theta(n)$ and the blowup-ratio is a constant, then the read complexity must be $\theta(n)$. In Chapter 3 it is shown that this lower bound is outperformed when considering an *adaptive* reader.

Chapter 3

Fault-Tolerant Adaptive Storage System

In this chapter we consider a different model than the one in Chapter 2. In this model the adversary is (almost) non-adaptive and the retrieval scheme makes its queries in a logarithmic number of adaptive rounds. For starters, consider a model in which the adversary can make no queries at all (fully non-adaptive) and the reader can query a single processor, before selecting a set to read from. In this model, same results as in the random-fault model can be achieved. The intuition is as follows: A secret permutation π on the processors is stored at each processor. Every write/read operation intended on i is applied to $\pi(i)$ (the reader knows π after he queries a single processor). Since the adversary, being non-adaptive, is unaware of π , the faults look random from the storage system point of view. This solution is simple but not very interesting as it immediately fails when the adversary is given the option to query a small number of processors before selecting the faulty set. Also, selecting and maintaining π requires a centralized solution we wish to avoid. However, this example shows that an advantage in the use of adaptive queries can help a lot.

In contrast, we present in this chapter a storage system which is fault-tolerant in a slightly enhanced adversarial model, where the adversary is allowed to have a small number of adaptive queries. The ‘secret’ which is kept from the adversary is only the location of files, no secret key is assumed in our solution. This keeps the system easy to implement in a dynamic and scalable environment.

The storage system we come up with has $O(\sqrt{n})$ write-complexity, $O(\sqrt{n} \log n)$ read-complexity and a constant blowup-ratio. It is resilient against $\theta(n)$ faults in the above model. Adaptive queries to a small set of processors enable finding a large fraction of the ones used for storage. It follows from Theorem 2.10 that this is not possible in the non-

adaptive retrieval model.

3.1 Small Load Implies Some Resiliency Against a Non-adaptive Adversary

We wish to upper bound the probability that a non-adaptive adversary succeeds in failing a set chosen by a write strategy. We show that as long as the load of a write strategy is small enough, a large fraction of a write set survives a non-adaptive attack, with some constant probability.

Suppose that a reader wishes to find a single live element of a write set, sampled from a write strategy μ_w with an almost optimal load of $O(\frac{1}{\sqrt{n}})$ [22]. It follows from the birthday paradox that a randomly chosen set of processors, of size $O(\frac{1}{\mathcal{L}(\mu_w)})$ intersects a previously sampled write set with some constant probability. The intersection is almost uniformly distributed over the elements of the write set. Theorem 3.1 shows that even after a non-adaptive adversary crashes some constant fraction of the network, with a constant probability at least half the elements of a write set survive. It follows from Lemma 2.8 that such a reader has an optimal read complexity, up to a constant.

Theorem 3.1. *Let U be a universe of processors of size n . Let μ be a write strategy on U with load \mathcal{L} and let $Q \sim \mu$. For every set $T \subseteq U$ of faulty processors,*

$$\mathbb{E}[|T \cap Q|] \leq |T| \cdot \mathcal{L} .$$

In the above setting, the probability that a constant fraction of a write set survives is

Corollary 3.2. $\Pr_{\mu}[|T \cap Q| > \lambda \mathbb{E}[|Q|]] < \frac{|T|\mathcal{L}}{\lambda \mathbb{E}[|Q|]} .$

Proof. Given $T \subseteq U$, let x_i be an indicator random variable that equals 1 if element i belongs to the intersection $Q \cap T$, and 0 otherwise. It follows that

$$\mathbb{E}[|T \cap Q|] = \mathbb{E}\left[\sum_{i=1}^n x_i\right] = \sum_{i=1}^n \mathbb{E}[x_i] = \sum_{i=1}^n \Pr[x_i = 1] = \sum_{i \in T} \Pr[x_i = 1] \leq \sum_{i \in T} \mathcal{L} = |T| \cdot \mathcal{L} . \quad (3.1)$$

Using (3.1) and the Markov inequality we upper bound the probability of the set Q to intersect T with more than $\lambda \mathbb{E}[|Q|]$ of its elements.

$$\Pr[|T \cap Q| > \lambda \mathbb{E}[|Q|]] = \Pr[|T \cap Q| > \frac{\lambda \mathbb{E}[|Q|]}{\mathbb{E}[|T \cap Q|]} \mathbb{E}[|T \cap Q|]] \leq \frac{\mathbb{E}[|T \cap Q|]}{\lambda \mathbb{E}[|Q|]} \leq \frac{|T| \cdot \mathcal{L}}{\lambda \mathbb{E}[|Q|]}$$

□

In particular, for a system with a fixed write set size \sqrt{n} , and write load $\frac{1}{\sqrt{n}}$ that faces up to δn faults, the probability that the majority of processors in a write set are crashed is upper bounded by 2δ .

3.2 A Generic Scheme for a Fault-Tolerant Storage System With a Constant Blowup Ratio

We propose a generic scheme with an adaptive retrieval algorithm which is fault-tolerant in the non-adaptive adversarial model. We saw that when the write load is close to optimal, then with a constant probability a constant fraction of a set chosen by the write strategy survives a non-adaptive attack. Suppose that an adaptive reader can find those surviving elements with a relatively small number of read operations. A pair of such a writer and reader is used to create a fault-tolerant storage system with a constant blowup ratio in the following way: A file f is encoded into ℓ pieces of data, such that any $\frac{\ell}{2}$ of them suffice for reconstructing f . An erasure coding scheme with a constant blowup ratio (e.g. Reed-Solomon) or an IDA scheme [25] is used for this purpose. These pieces of data are then distributed to the ℓ members of a selected write set.

Now, let a non-adaptive adversary crash a δ fraction of the processors in the network, where $0 \leq \delta < \frac{1}{2}$. Corollary 3.2 implies that a write set chosen by an optimal load strategy, has a probability of at least $1 - 2\delta$ that half of the write set survived. An adaptive reader can then identify the surviving members and reconstruct the file. To ensure reconstruction with an arbitrarily high probability it suffices to write to a constant number of write sets, chosen independently.

The pair of write and read strategies of the ϵ -intersecting QS [16] does not suit this scheme. While having a small write load, a reader must read a constant fraction of the network in order to find a large intersection with a write set (this fact is what makes it resilient against an adaptive adversary). The paths QS presented in [22] was shown to have an optimal load. The elements of a single path can be found with $O(\sqrt{n})$ queries in $O(\sqrt{n})$ rounds, in the presence of faults. Thus, it can be used to create a fault-tolerant storage system with $O(\sqrt{n})$ read and write complexity and a constant blowup ratio. Next we show a write strategy with complexity $O(\sqrt{n})$ and an adaptive reader with complexity $O(\sqrt{n} \log n)$ that requires only $O(\log n)$ rounds.

In this model, when a storage system is $\theta(n)$ -fault-tolerant, we cannot expect the product of the read complexity and the write complexity to be $o(n)$. This is due to the fact that adaptiveness does not help before the first element is found. Thus, Theorem 2.9 can be used

to show this lower bound.

3.3 Adaptive Storage System Based on the ‘And-Or’ System

We present a pair of write strategy and adaptive read algorithm, based on the ‘And-Or’ quorum system shown in [22] which has an optimal load of $O(\frac{1}{\sqrt{n}})$. The quorum size is \sqrt{n} , and the read algorithm finds a write set in $O(\log n)$ rounds, using $O(\sqrt{n} \log n)$ queries.

We recall the construction of the ‘And-Or’ system. Consider a complete binary tree of height h , rooted at $root$, and identify the 2^h leaves of the tree with systems processors. We define two collections of subsets of the set of processors, using the following recursive definitions:

- (i) For a leaf v , $ANDset(v) = ORset(v) = \{\{v\}\}$.
- (ii) $ANDset(v) = \{S \cup R \mid S \in ORset(v.left) \wedge R \in ORset(v.right)\}$.
- (iii) $ORset(v) = ANDset(v.left) \cup ANDset(v.right)$

The And-Or quorum system is then composed of the collections $ANDset(root)$ and $ORset(root)$. A natural recursive procedure can be used to generate a set $S \in ANDset(root)$. The procedure visits nodes of the tree, beginning at the root and propagating downwards. It considers the nodes of the tree as AND/OR gates (on even/odd levels). When visiting an AND gate it continues to both its children, and when visiting an OR gate one of its children is chosen. The leaves visited by the procedure form S . A similar procedure generates a set $R \in ORset(root)$. The And-Or structure induces the following properties:

Lemma 3.3 (from [22]). *Consider a complete binary tree of height h rooted at $root$. Let $S \in ANDset(root)$ and $R \in ORset(root)$ then $|S \cap R| = 1$, $|R| = 2^{\lfloor \frac{h}{2} \rfloor}$ and $|S| = 2^{\lfloor \frac{h+1}{2} \rfloor}$.*

3.3.1 Adaptive Read Strategy for Finding a Complete And Set.

We assume a writer uniformly picks a set $S \in ANDset(root)$ to serve as a write set. A uniform sample can be obtained by choosing *left* or *right* with equal probability, in every OR gate during the activation of the recursive procedure. A reader wishes to identify all the members of S .

By probing a processor, the reader finds out whether it belongs to S or not. A single element of S can be identified by probing the $O(\sqrt{n})$ members of an arbitrary set $R \in$

$\text{ORset}(root)$, as follows from Lemma 3.3. After finding a single processor $s \in S$, a large fraction of the original tree can be pruned, as s reveals the choices made at the OR gates on the path from $root$ to s . After pruning, the remainder of the tree is composed of complete binary trees of heights $0, 2, 4, \dots, \log n - 2$. The algorithm is then recursively applied. Figure 3.1 illustrates a pruned tree after a single element was found. The recursive Algorithms 3.1 and 3.2 describe the process of identifying the elements S more formally.

Algorithm - Identify ANDset on AND Gate

Input: A root $root$ of And-Or tree (AND gate)

Output: A subset of leaves contained in S .

1. Uniformly choose a set $R \in \text{ORset}(root)$. Probe the elements of R .
2. If the intersection $R \cap S$ is empty then finish. Else, set $s := R \cap S$ be the element in the intersection of R and S .
3. Let $(root = s_0, s_1, \dots, s_k = s)$ denote the path from $root$ to s on the And-Or tree.
4. For each even s_i on even i (AND gate):
 - (a) If $s_{i+1} = s_i \rightarrow left$ then set $T_{\frac{1}{2}i} := s_i \rightarrow right$, else set $T_{\frac{1}{2}i} := s_i \rightarrow left$
 - (b) Assign $S_{\frac{1}{2}i} := \text{IdentifyANDsetORGate}(T_{\frac{1}{2}i})$
5. return $\{s\} \cup \bigcup_{i=0}^{\frac{1}{2}k} S_i$

Table 3.1: Algorithm Identify ANDset on AND Gate.

The correctness of the algorithm is stated in the following Lemma:

Lemma 3.4. *Let $S' := \text{IdentifyANDsetORGate}(root)$ be the set returned by algorithm 3.1, then $S' = S$.*

Proof. It is clear that $S' \subset S$ because only elements of S are ever inserted into S' . We prove now the opposite direction i.e., $S \subset S'$. The proof is by induction on h , the height of the tree. The induction base is a tree v of height 0 (a single leaf). In this case an $\text{ANDset}(v)$ is v itself, and it will be returned in step 2 of Algorithm 3.1 as the one element of $\text{ORset}(root)$. The induction hypothesis is that if $S \in \text{ANDset}(root)$, on a tree of height h , then $S' = S$

Let T be a tree of height $h + 1$, rooted at $root$, and let $S \in \text{ANDset}(root)$ on this tree. Let S' be the set returned in step 5 of Algorithm 3.1 applied to $root$. Consider an element

Algorithm - Identify ANDset on OR Gate**Input:** A root $root$ of And-Or tree (AND gate)**Output:** A subset of leaves contained in S .

1. if $root$ is a leaf, return $root$,
 2. else,
 - (a) Set $S_1 := \text{IdentifyANDsetORGate}(root \rightarrow left)$
 - (b) Set $S_2 := \text{IdentifyANDsetORGate}(root \rightarrow right)$
 3. return $S_1 \cup S_2$
-

Table 3.2: Algorithm Identify ANDset on OR Gate.

$s \in S$. If s is the element in the intersection with the set R , probed in step 1 then $s \in S'$. Assume this is not the case, then s must be a leaf in one of the subtrees T_i of step 4a. By the induction hypothesis, s is then found in one of the recursive calls of step 4b, because each such call generates a call to Algorithm 3.1 (in Algorithm 3.2, step 2) on a tree of height $h - 1$ or less. \square

Proposition 3.5. *The read complexity of Algorithm 3.1, for identifying all the members of $S \in \text{ANDset}(root)$, where $root$ is the root of a tree of height h , is $O(\sqrt{n} \log n)$, where $n = 2^h$ is the number of leaves in the tree. The number of rounds required for the process is $\log n$.*

Proof. Let $f(h)$ represent the number of probes done by Algorithm 3.1 when it acts on a tree of height h , in case step 2 resulted in finding an element in the intersection. Let $f'(h)$ represent the number of probes done by this algorithm in the other case. Respectively, let $g(h)$ represent the number of probes done by Algorithm 3.2. Processors are probed in Algorithm 3.1 in step 1 and in the recursive call in step 4b. The number of probes in the case an intersecting element was found is therefore

$$f(h) = 2^{\frac{h}{2}} + \sum_{j=1}^{\frac{h}{2}} g(2j - 1), \quad (3.2)$$

In the case no element was found in the intersection, we simply count the size of R :

$$f'(h) = 2^{\frac{h}{2}}. \quad (3.3)$$

When $h = 0$ $f(0) = 1$.

The probes done in Algorithm 3.2 called on a height h tree, results from two recursive calls to Algorithm 3.1 on a trees of height $h - 1$. One of these calls will result in finding an intersecting element in step 2 of Algorithm 3.1 and the other will result in no intersection at this stage (because we are at an OR gate, and only one of it children was followed). Therefore,

$$g(h) = f'(h - 1) + f(h - 1) = 2^{\frac{h-1}{2}} + f(h - 1). \quad (3.4)$$

Next, we show that $f(h) \leq ch2^{\frac{h}{2}}$ for some constant $c > 1$. We do so by induction. For $h = 0$, $f(h) = 1$. We assume that for $h \geq 1$, $f(h) \leq c \cdot h2^{\frac{h}{2}}$ and show that $f(h + 2) \leq c(h + 2)2^{\frac{h+2}{2}}$:

$$f(h + 2) = 2^{\frac{h+2}{2}} + \sum_{j=1}^{\frac{h+2}{2}} g(2j - 1) \quad (3.5)$$

$$= 2^{\frac{h+2}{2}} + \sum_{j=0}^{\frac{h}{2}} 2^j + f(2j) \quad (3.6)$$

$$\leq 2 \cdot 2^{\frac{h+2}{2}} + \sum_{j=0}^{\frac{h}{2}} f(2j) \quad (3.7)$$

$$\leq 2 \cdot 2^{\frac{h+2}{2}} + \sum_{j=0}^{\frac{h}{2}} 2cj2^j \quad (3.8)$$

$$\leq 2 \cdot 2^{\frac{h+2}{2}} + ch \sum_{j=0}^{\frac{h}{2}} 2^j \quad (3.9)$$

$$\leq 2 \cdot 2^{\frac{h+2}{2}} + ch2^{\frac{h+2}{2}} \leq c(h + 2) \cdot 2^{\frac{h+2}{2}} \quad (3.10)$$

where the second inequality derives from the induction hypothesis. Finally, assigning $h = \log n$ completes the proof. \square

3.3.2 Adaptive Read Strategy in the Presence of Faults.

We now assume the existence of up to δn crash faults in the network caused by a non-adaptive adversary, where $0 < \delta < \frac{1}{2}$. Let $S \in \text{ANDset}(\text{root})$ be chosen at random as described earlier. Corollary 3.2 implies that with a constant probability at least half the elements of S are not

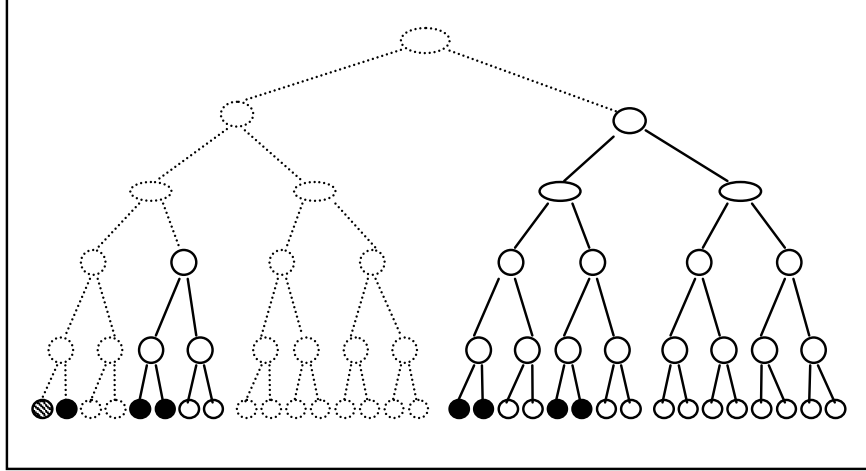


Figure 3.1: The colored leaves are the set S chosen from the collection $\text{ANDset}(\text{root})$. After the left most leaf was identified as belong to S , the tree can be pruned as demonstrated

deleted. The reader's algorithm is now required to identify those surviving elements. In this case, finding an element of the chosen S , cannot be guaranteed by querying an arbitrary set in $\text{ORset}(\text{root})$, as the element in the intersection may be crashed. To overcome this, we may randomly probe processors for membership. To guarantee success with an arbitrarily high probability, it suffices to query $O(\sqrt{n})$ elements as discussed in Section 3.1. The read complexity of the new algorithm remains $O(\sqrt{n} \log n)$. Combining the above write strategy and the adaptive read algorithm, with the generic scheme of Section 3.2, yields the following theorem.

Theorem 3.6. *The system described above is a $\theta(n)$ -fault-tolerant, adaptive storage system, with a constant blowup-ratio, $O(\sqrt{n})$ write complexity and $O(\sqrt{n} \log n)$ read complexity. The number of adaptive rounds is $\log(n)$.*

Theorem 3.6 considers the fault-tolerance of the described storage system in the (fully) non-adaptive adversarial model. The fault-tolerance properties of the system are preserved even in the presence of a much stronger adversary with the power to query a $o(\sqrt{n})$ processors prior to the selection of the set to crash. Theorem 3.6 remains correct under this ‘almost non-adaptive’ adversarial model. The proof relies on the fact that by querying $o(\sqrt{n})$ processors, an adversary is unlikely to identify more than a constant number of elements of a write set.

3.3.3 Dynamically Adjusting the Retrieval Scheme to the Number of Faults.

A drawback of the And-Or storage system, is that even in the case when much fewer than $\theta(n)$ faults occur the read complexity cannot be reduced. In [3] Alvisi et al show how a Byzantine quorum system dynamically adjusts to the actual number of faulty processors. It allows them to operate with relatively small quorums in the absence of faults, increasing the quorum size as faults appear. Our system can also be improved so that it reads significantly fewer elements, when there are fewer than $\theta(n)$ faults. The cost of this feature is a $O(\log n)$ blowup-ratio and a multiplicative constant to the write complexity. Unlike [3], no explicit knowledge of the number of faults is required.

The following modifications to the And-Or structure are required: First, each node in the tree represents a processor (not only the leaves). Second, the definitions of the ANDset and ORset collections of sets are changed so as to include all nodes visited by the recursive procedure that generates a set in these collections.

- (i) For a leaf v , $\text{ANDset}'(v) = \text{ORset}'(v) = \{\{v\}\}$.
- (ii) $\text{ANDset}'(v) = \{\{v\} \cup R \cup S \mid R \in \text{ORset}'(v.\textit{left}) \wedge S \in \text{ORset}'(v.\textit{right})\}$.
- (iii) $\text{ORset}'(v) = \{\{v\} \cup S \mid S \in \text{ANDset}'(v.\textit{left}) \cup \text{ANDset}'(v.\textit{right})\}$

Figure 3.2 illustrates a set in the collection $\text{ANDset}'(\textit{root})$.

Let $S \in \text{ANDset}'(\textit{root})$. We note that the size of S does not change by more than a multiplicative constant. Let S_i denote the members of S in the i th level of the tree. For each i , S_i can be thought of as a set belonging to $\text{ANDset}(\textit{root})$ defined on a tree with i levels. The write process encodes a file separately into the set S_i for each i as in the original scheme. This way, reading from the members of a single set S_i (for each i) suffices to reconstruct the file. We therefore achieve $O(\log n)$ blowup-ratio and $O(\sqrt{n})$ write complexity. During retrieval we seek the sets S_0, \dots, S_h in an increasing order. The members of a set S_i can be identified using the algorithm described in Section 3.3.2 applied on the tree of height i . The retrieval process stops when a set with a sufficient number of surviving elements is found. It follows from Theorem 3.6 that when there are g faults, such a set will be found at level of height $\lceil \log g \rceil$ i.e, enough members of $S_{\lceil \log g \rceil}$ survive and can be found. Hence the read complexity is $O(\sqrt{g} \log g)$.

Theorem 3.7. *The system described above is a $\theta(n)$ -fault-tolerant, adaptive storage system, with $O(\log n)$ blowup-ratio, $O(\sqrt{n})$ write complexity and a read complexity that does not exceed $O(\sqrt{g} \log g)$ when up to g faults occur.*

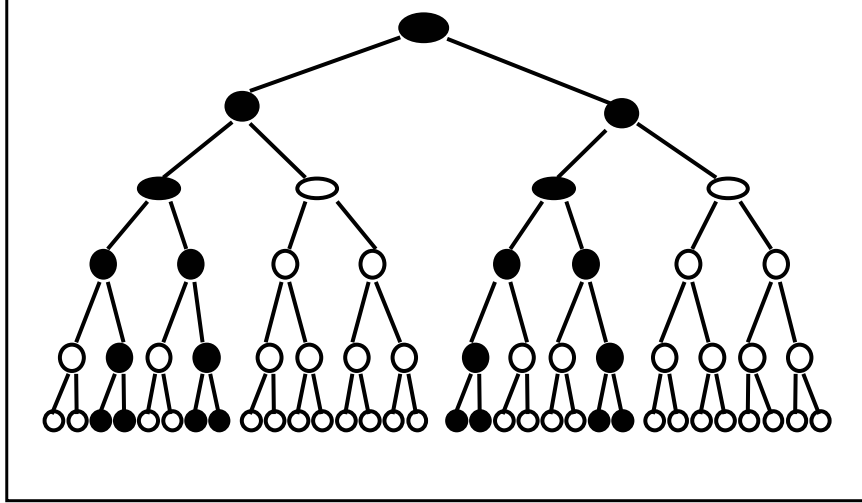


Figure 3.2: The black nodes represent a set chosen from $\text{ANDset}'(\text{root})$. Each node in the tree represents a processor and not only the leaves.

Proof. We first show that the size of a write set is $O(\sqrt{n})$: Let $\text{ANDset}(\text{root}, i)$ denote the ANDset structure defined on a tree of height i . A set $S' \in \text{ANDset}'(\text{root})$, can be decomposed into h disjoint sets, S_1, \dots, S_h , such that $S_i \in \text{ANDset}(\text{root}, i)$. Lemma 3.3 states that for each $1 \leq i \leq h$, $|S_i| \leq 2^{\frac{i+1}{2}}$. Thus,

$$|\text{ANDset}'(\text{root})| \leq \sum_{i=1}^h 2^{\frac{i+1}{2}} = \sum_{i=1}^{\frac{h}{2}} 2^i + \sqrt{2} \sum_{i=1}^{\frac{h}{2}} 2^i = (1+\sqrt{2}) \cdot 2^{\frac{h}{2}+1} \leq (1+\sqrt{2})(|\text{ANDset}(\text{root})|)$$

And so the write complexity does not grow by more than a multiplicative factor.

From Theorem 3.6 and the above decomposition of a set S' it is clear that the system can tolerate $\theta(n)$ adversarial faults. The read algorithm for the new system will operate in stages. In each stage it will consider only the processors in the i' th level and will look for a write set in this level using Algorithm 3.1. The read complexity of the new system is then $O(\sqrt{n} \log n)$. \square

Chapter 4

Dynamic Implementation of the And-Or Storage System

In this chapter we present an adaption of the And-Or storage system presented in Chapter 3, to a dynamic environment, where processors may join and leave the system. For this adaptation we make use of the notion of Distributed Hash Table (DHT). Specifically, we suggest an implementation over the 'Distance Halving' overlay network, presented by Naor and Wieder [19].

A distributed hash table is a distributed system in which data items (or files) are associated with a key, and each processor in the network is responsible for a certain range of keys. In a dynamic environment, the mapping from the range of keys onto the set of processors is constantly being updated as processors join and leave the network. Among recently proposed DHT constructions for a dynamic environment are CAN [26], Chord [29], Pastry [27], Tapestry [31], and Viceroy [13]. The mapping of files to nodes is done in the same manner in all of these constructions: The range of keys, denoted I is entirely decomposed among all active processors in the network. Files are mapped into I using a hash function h , i.e., a file f is mapped into $h(f.name)$. Each processor stores the files which are mapped into keys in the range which it covers.

The key idea behind the dynamic construction of the And-Or storage system, is to keep an embedding of binary trees over the active processors. To achieve a file system functionality, an And-Or tree is defined for each file that can potentially be stored in the system. The processor that covers $h(f.name)$ now serves as the root of such a binary And-Or tree. Notice that a single And-Or tree over the processors would basically suffice for constructing the And-Or storage system described in Chapter 3. However, the solution for static networks, did not take into account the cost of communication with storing processors. The cost of routing

is essential measure for the quality of a storage system in a dynamic environment, where processors are generally loosely coupled. In our solution, also internal nodes, as well as leaves, represent processors. The internal nodes are used in the routing scheme of messages of the storage system. Multiple And-Or trees are defined to achieve a low load on processors during the routing process of the storage system messages. Consequently, a generic And-Or storage system may be implemented over an overlay network with the following properties:

1. A key in the range I can be easily located, i.e., the processor that covers a key can be located in sublinear time.
2. Each processor has (at least conceptually) two children defined. This way, each processor can serve as the root of a binary tree over the processors.
3. The number of different processors that serve as leaves in such trees, in depth $\log n$ is $\theta(n)$. (where n is the current number of active processors).

A network topology for which conditions 2 and 3 holds is the De-Bruijn graph. The d -dimensional De-Bruijn graph consists of 2^d processors and 2^{d+1} directed edges. Each node corresponds to a d -bit binary string. There is a directed edge from each node $u_1u_2 \cdots u_d$ to $u_2 \cdots u_d0$ and $u_2 \cdots u_d1$. Several dynamic constructions emulate the De-Bruijn network topology, among are the DH-network, Koorde [7] and the Overlay network [1].

We chose the DH-network as an overlay network for the dynamic storage system, mainly because of the continuous-discrete approach taken in the DH construction. In this approach, a continuous graph is defined over the interval, such that each point $x \in I$ has two children and serves as a root of a binary tree subgraph. Embedding a binary tree becomes natural for this scheme. Conditions 2 and 3 follows immediately from the definition of the continuous graph. Condition 1 is achieved using the routing scheme of the DH-network. This approach, in which the binary tree edges are defined over the continuous graph and later emulated using the network connection enable data migration through a gossip protocol, makes the maintenance of the storage system built in the DH-network, and keeps the analysis of the protocols much simpler. In Section 4.1 we describe the DH-network. In Section 4.2 we describe in details, the dynamic And-Or storage system.

4.1 Some Properties of the Distance Halving Dynamic Network

We recall the construction of the DH-network from [19]. We first define the DH-graph $G_c = (E_c, V_c)$. The vertex set of G_c is the unit interval $I \triangleq [0, 1)$. The edge set of G_c is defined by the following functions:

$$\ell(y) \triangleq \frac{y}{2}, \quad r(y) \triangleq \frac{y+1}{2},$$

where $y \in I$, and ℓ and r abbreviates ‘left’ and ‘right’ respectively. The edge set of G_c is then

$$E_c \triangleq \{(y, \ell(y)), (y, r(y)) \mid y \in I\}.$$

We denote by $parent(x)$ the function that returns the parent of a point $x \in I$ in the DH-graph i.e., $parent(x) = \begin{cases} 2x, & x < \frac{1}{2}, \\ 1 - 2x, & x \geq \frac{1}{2}. \end{cases}$

Denote by \vec{x} a set of n points in I . The point x_i serves as the identity of the i th processor, in a network of n processors. The points of \vec{x} divide I into n segments. Define the segment of x_i to be $s(x_i) = [x_i, x_{i+1})$ ($i = 1, \dots, n-1$) and $s(x_n) = [x_n, 1) \cup [0, x_1)$. In the DH-network, each processor u_i is associated with the segment $s(x_i)$. If a point y is in $s(x_i)$ we say that u_i covers y . Notice that each point $y \in I$ is covered by exactly one processor. When data is written to y it is being stored in the processor that covers y , and remains stored in a single processor throughout the lifetime of the DH-network (when no faults occur). A pair of processors (u_i, u_j) are connected if there exists an edge (y, z) in the DH-graph, such that $y \in s(x_i)$ and $z \in s(x_j)$. The edges in the continuous and discrete DH-graphs are illustrated in Figure 4.1.

To implement the And-Or system on a DH-network we identify an embedding of a binary tree on the DH-graph. The $\ell(\cdot)$ and $r(\cdot)$ functions induce an embedding of a complete binary tree on every point x in I , in a natural way. The left child of a point x is $\ell(x)$ and the right child is $r(x)$. We observe that the distance between every two adjacent points, that represent leaves on the same level ℓ is exactly $\frac{1}{2^\ell}$. Note however, that the distance between two points that represent the two direct children of the same node, is $\frac{1}{2}$. We note that the same tree structure was used in [19] to relieve ‘hot spots’ (points with high load).

As is common in DHT literature, we make the simplifying assumption that an underlying layer takes care of concurrent operations of Join/Leave/Read/Write, so that the DH-network implementation gets these operations sequentially. For the consistency and correctness of the DH DHT it is assumed that the operations of join/leave/read/write do not overlap. We

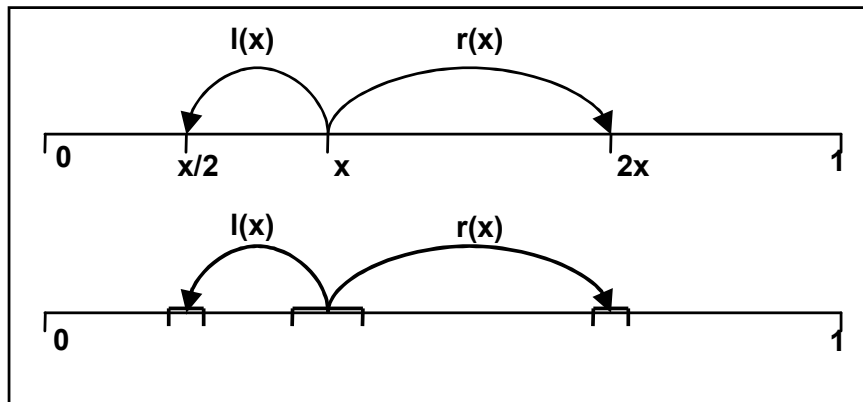


Figure 4.1: Each point x in the distance halving graph is connected with a directed edge to $\frac{x}{2}$ and to $2x$. Down: Two segments are connected with an edge in the discrete graph, if they have edges between them in the continuous graph.

refer the reader to [12] where techniques for achieving concurrency in peer-to-peer networks are introduced.

4.2 Dynamic fault-tolerant Scheme

We show an implementation of a dynamic storage system, based on the And-Or system over the DH-network. In [19] a file f was stored at a processor that covered $h(f.name)$, where h is a hash function mapping from the domain of file names into I . We use $h(f.name)$ as the root of the And-Or tree. Let $\tau(f)$ denote the tree of height $\log n$ rooted at $root \triangleq h(f.name)$. When f is distributed to a randomly chosen $S \in \text{ANDset}(root)$, it is actually stored in the processors that cover the members of S .

When no δ fraction of the processors cover too large a fraction of I for small enough δ , the system is $\theta(n)$ -fault tolerant, because then enough leaves survive an attack. This condition is satisfied when the DH-network is balanced, i.e. no processor cover a segment of size more than $\frac{c}{n}$, for some constant c . Various balancing techniques are described in [19, 13, 8]. Manku presented in [17] join/depart operations in a dynamic network that maintain a maximum ratio of 4 between the size of intervals covered by every two processors.

Theorem 4.1. *Assume the DH-network is balanced, then the above storage system is $\theta(n)$ -fault tolerant.*

Proof. Theorem 3.6 states that the storage system described in Chapter 3 is fault-tolerant

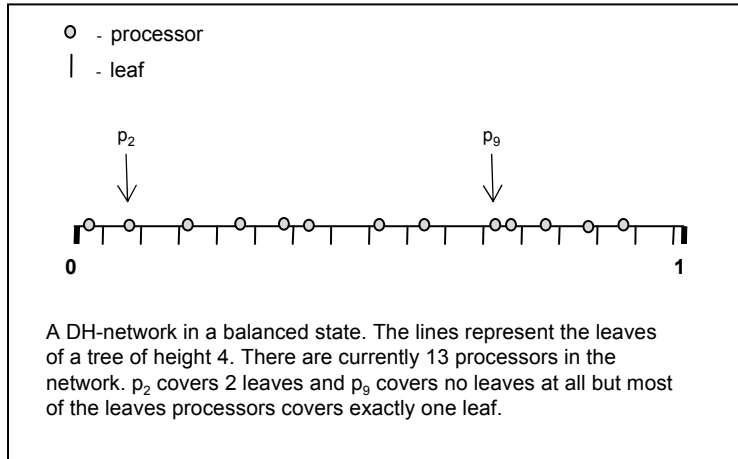


Figure 4.2: Balanced Network.

for some constant $\delta > 0$. There each server corresponds to a different single leaf in the And-Or tree. In the dynamic solution when the system is kept balanced, a single processor may cover up to c leaves, for some constant c . In this case, an adversary that can crash up to $\delta' = \frac{\delta}{c}$ fraction of the processors will not delete more than a delta fraction of the leaves of any And-Or tree defined on I , of height $\log n$. Thus, the dynamic system is $\delta'n$ fault-tolerant. \square

We note that when the identities are uniformly chosen in the join process, the system remains $\theta(n)$ -fault tolerant. Uniform choice does not achieve ‘balancing’ since with high probability there will be a processor that covers a segment of size $\theta(\frac{\log n}{n})$. However it remains guaranteed that for small enough δ , no set of δn processors covers too large a fraction of I . Thus, adversary cannot control too many leaves.

4.2.1 Network Implementation of Storage and Retrieval

Storage Through Gossip

A naive way to store encoded strings in the processors that cover leaves is in a direct way, by using the original routing of the DH-network ([19]). An alternative, which was already mentioned in the generic construction at the beginning of this Chapter, is to write all the ‘write data’ to a single processor, and let it percolate to the leaves with a ‘gossip protocol’. The write data is composed with all the encoded pieces of data that would eventually be

stored at leaves. As a first step in storing a file, the write date is written to the root of $\tau(f)$. It then, in logarithmic number of steps, percolates from the root of $\tau(f)$ until reaching the set of write processors S . On even steps (representing AND gates in the And-Or tree) the data is equally divided into two parts, each are then ‘gossiped’ to a different child. On odd steps (OR gates), the data is sent to one of the children chosen at random. After $\log n$ steps the data reaches the members of S which is a randomly chosen set from the collection $\text{ANDset}(\text{root})$.

From the writer’s point of view the process concludes after writing to the root. At this point the file can already be retrieved. Fault-tolerance is acquired during the percolation: After the gossip has reached at least the nodes of level i , the file becomes $\Omega(2^i)$ -fault-tolerant. By letting each processor keep a copy of the data it got during the percolation phase, the system becomes dynamically adjusted to number of faults, as discussed in Section 3.3.3.

The gossip protocol also saves in message complexity in comparison with the alternative of storing the data in each leaf separately. In the gossip protocol, the message complexity would be $\theta(\sqrt{n})$, whereas the naive approach would require routing \sqrt{n} messages, paying an average of $\log n$ routing hops per message.

Retrieval

Retrieval is done as in the And-Or system using the routing protocol of the DH-network. Routing dilation is $O(\log n)$, hence retrieval takes $O(\log^2 n)$ steps.

Both storage and retrieval protocols use the value of $\log n$. Usually, the size of the network n is not available to processors in a dynamic network, so some approximation is required. Such an approximation is suggested in [13] as part of the Viceroy network. The same technique can also be used in the DH-network.

In Chapter 6 we discuss in more details, similar gossip protocols, for a dynamic quorum system, based on the same embedding of the And-Or tree, over the DH-network.

Chapter 5

The And-Or Quorum System In a Random Faults Model

5.1 Motivation and Definitions

In the first chapters we have discussed the fault-tolerance of storage systems in an adversarial model. We now shift to the random-fault model. We assume that processors fail independently with probability p . We assume also that the failures are crash failures and that they are detectable.

Definition 5.1. *A configuration is a vector $\mathbf{x} \in \{0, 1\}^n$ in which $x_i = 1$ if and only if (iff) processor $i \in U$ has failed.*

For a configuration \mathbf{x} let $\text{dead}(\mathbf{x}) = \{i \in U : x_i = 1\}$ denote the set of failed elements, and let $\text{live}(\mathbf{x}) = \{i \in U : x_i = 0\}$ denote the set of functioning elements.

We formally repeat here the definition of the global failure probability [23] mentioned in Chapter 1, which measures the availability of the system.

Definition 5.2. *For every quorum Q in a quorum system \mathcal{Q} let \mathcal{E}_Q be the event that Q is hit, i.e., at least one element of Q has failed. Let $\text{fail}(\mathcal{Q})$ be the event that all quorums $Q \in \mathcal{Q}$ are hit, i.e., $\text{fail}(\mathcal{Q}) = \bigcup_{Q \in \mathcal{Q}} \mathcal{E}_Q$*

Definition 5.3. *The global failure probability $F_p = \Pr_p[\text{fail}(\mathcal{Q})]$, where the probability space is the product distribution over all possible configurations.*

Algorithmic Probe Complexity.

The availability measure tells us about the probability of a quorum set to survive in the random-fault model, but tells us nothing about the complexity of finding a live quorum. In [24], Peleg and Wool defined the probe complexity of quorum systems and analyzed it for some known quorum systems. It is assumed that an adversary decides which processors fail and the probe complexity is defined as the number of probes required before a live quorum set is found or an evidence for the lack of it. A probe action is assumed to have $O(1)$ complexity. In [6] Hassin and Peleg analyzed the probe complexity of several systems when in their fault-model each processor independently fails with some fixed probability. In [20] Naor and Wieder defined the *Algorithmic probe complexity* as the actual time and message complexity required for finding a live quorum. We use their definition in this work.

An algorithm for finding a live quorum which decides on which processors to probe prior to gaining any knowledge as to which processors failed is called *non-adaptive*. A lower bound on the algorithmic probe complexity of a non-adaptive algorithm, as a function of the load was given in [20]:

Theorem 5.4 (From [20]). *Let \mathcal{S} be a quorum system over a universe of processors U with load $\mathcal{L} = \mathcal{L}(\mathcal{S})$. Assume that each element in U fails with some fixed probability $p \leq \frac{1}{2}$. Let $X \subset U$ be a predefined set of elements s.t.,*

$$\Pr[X \text{ contains a live quorum}] \geq \frac{1}{2},$$

then

$$|X| \geq \frac{1}{2 \log(1/p) + 1} \frac{\log(1/4\mathcal{L})}{\mathcal{L}}.$$

In particular if $\mathcal{L} = O(\frac{1}{\sqrt{n}})$ then $|X| = \Omega(\sqrt{n} \log n)$.

An *adaptive* algorithm can act in a number of rounds, such that in each round the probe results of prior rounds are known. We present now non-adaptive and adaptive algorithms for finding a live quorum for the And-Or system.

5.2 Algorithmic Probe Complexity of the And-Or Quorum System.

In Chapter 3 we described the $\text{ANDset}(root)$ and $\text{ORset}(root)$ collections of sets, on a binary tree of height h , rooted at $root$. For ease of reading we repeat their definition here:

- (i) For a leaf v , $\text{ANDset}(v) = \text{ORset}(v) = \{\{v\}\}$.
- (ii) $\text{ANDset}(v) = \{S \cup R \mid S \in \text{ORset}(v.\text{left}) \wedge R \in \text{ORset}(v.\text{right})\}$.
- (iii) $\text{ORset}(v) = \text{ANDset}(v.\text{left}) \cup \text{ANDset}(v.\text{right})$

Lemma 3.3 stated that each set $S \in \text{ANDset}(\text{root})$ intersects each set $R \in \text{ORset}(\text{root})$. This property is used to define the ‘And-Or’ quorum system: A quorum in the And-Or system is any set $Q = S \cup R$ where S is a member of $\text{ANDset}(\text{root})$ and R is a member of $\text{ORset}(\text{root})$. For a binary tree T we denote the set of quorums with $\text{AndOr}(T)$. The load of the And-Or system was shown to be optimal ($O(\frac{1}{\sqrt{n}})$) in [22].

The And-Or system was shown to have a failure probability F_p which exponentially decays in n . For $p \leq \frac{1}{4}$ it was shown in [22] that $F_p(\text{AndOr}) \leq \exp(-\Omega(\sqrt{n}))$. For later analysis we wish to quote a lemma from [22] that states the failure probability of the $\text{ANDset}(\text{root})$ and $\text{ORset}(\text{root})$ collections:

Lemma 5.5 (from [22]). *Let $F_A(h)$ denote the probability that all sets in the $\text{ANDset}(\text{root})$ collection are hit, in a complete binary tree rooted at root , of height h . Similarly let $F_O(h)$ denote the failure probability of the $\text{ORset}(\text{root})$ collection. Then*

$$F_A(h) \leq (2p)^{2^{\frac{h}{2}}}, \quad F_O(h) \leq (4p)^{2^{\frac{h}{2}}}.$$

5.2.1 A Non-adaptive Algorithm.

We now show a non-adaptive algorithm for choosing a live quorum in the And-Or quorum system. The probe complexity of algorithm is $O(\sqrt{n} \log n)$ which matches the lower bound of Theorem 5.4.

Let r be the root of a complete binary tree. We introduce the notion of ANDset collection on internal nodes of the tree, in level h , denoted $\text{ANDset}(r, h)$:

Definition 5.6 ($\text{ANDset}(r, h)$). *Let T be a binary tree rooted at r with h levels. Let T' be the same tree, where all the nodes in level $h + 1$ or above are truncated, and the nodes of level h serve as leaves. Then $\text{ANDset}(r, h)$ equals the collection $\text{ANDset}(r)$ as defined on T' .*

The non-adaptive algorithm uniformly picks a set $R \in \text{ANDset}(r, h)$ where

$$h = \lfloor \log n - 2 \log \log n \rfloor.$$

All processors which are descendants of nodes in R are then chosen. Figure 5.1 illustrates a set chosen by the non-adaptive algorithm.

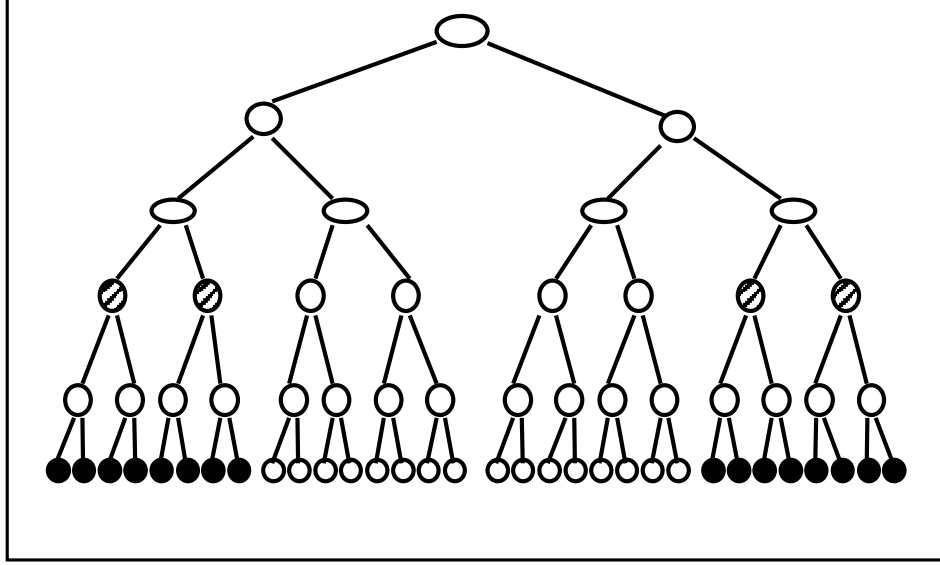


Figure 5.1: The semi-marked nodes at level 3 were chosen as a set in the collection $\text{ANDset}(\text{root}, 3)$. The black nodes form the non-adaptive set.

Lemma 5.7. *The cardinality of a set of processors S chosen by the non-adaptive algorithm is $O(\sqrt{n} \log n)$.*

Proof. Let $R \in \text{ANDset}(r, h)$, then by Lemma 3.3

$$|R| = O(2^{\frac{h}{2}}) = O\left(\frac{\sqrt{n}}{\log n}\right). \quad (5.1)$$

The number of processors which are descendants of an element in R is

$$2^{\lceil 2 \log \log n \rceil} = O(\log^2 n) \quad (5.2)$$

The lemma then follows from equations 5.1 and 5.2. \square

Observation 5.8. *Let S be a member of $\text{ANDset}(r)$ then there exists a set $R \in \text{ANDset}(r, h)$ and a collection $\{S_v : v \in R, S_v \in \text{ANDset}(v)\}$ such that:*

$$S = \cup_{v \in R} S_v.$$

Proof. This observation follows directly from the definition of the recursive procedure for selecting a member of $\text{ANDset}(r)$ \square

Theorem 5.9 (Non-adaptive Probe Complexity). *Let S denote the set of processors chosen as described above, then S contains a live set from the collection $\text{ANDset}(r)$ with high probability¹.*

Proof. Denote by \mathcal{E} the event that S contains a live member of $\text{ANDset}(r)$. It follows from Observation 5.8 that \mathcal{E} occurs iff a live set $S_v \in \text{ANDset}(v)$ exists for every $v \in R$. Denote by \mathcal{E}_v these later events (the non existence of a live S_v). From Lemma 5.5 it follows that

$$\Pr[\mathcal{E}_v] \leq 2^{-\lceil 2 \log \log n \rceil} \leq \frac{1}{n}.$$

The events \mathcal{E}_v are mutually independent, so:

$$\Pr[\mathcal{E}] = \Pr\left[\bigwedge_{v \in R} \overline{\mathcal{E}_v}\right] \geq \left(1 - \frac{1}{n}\right)^{O\left(\frac{\sqrt{n}}{\log n}\right)} \geq e^{-O\left(\frac{1}{\sqrt{n}}\right)} \geq 1 - O\left(\frac{1}{\sqrt{n}}\right),$$

This concludes the proof. □

In a similar way a subset of processors that contains a live member of $\text{ORset}(r)$ can be chosen. Joining a such a set with a live member of $\text{ANDset}(r)$ yields a live quorum in the And-Or quorum system.

5.2.2 An Adaptive Algorithm

We present an adaptive algorithm for finding a live quorum for the And-Or system which requires $\theta(\sqrt{n})$ probes. By an adaptive algorithm we mean an algorithm that probes several subsets of processors, in a number of rounds. The reader is assumed to send all queries in a specific round prior to getting any reply from any of the processors that are probed in this round. Answers from probed processors are returned before some timeout, after which the reader decides which processors to probe in the next round. The number of rounds therefore determines the time complexity of the algorithm so it is desired to have as small as possible number of rounds.

Our algorithm requires w.h.p. only $O(\log \log n)$ rounds. Naor and Wieder presented in [20] an adaptive algorithm for the Paths quorum system with a $\theta(\sqrt{n})$ probe complexity, however, their solution requires $\theta(\sqrt{n})$ rounds.

As in the non-adaptive case, it is sufficient to show how to find a live member of $\text{ANDset}(root)$. A live member of $\text{ORset}(root)$ can be found in a similar way. To adaptively find a live member of $\text{ANDset}(root)$, a reader first picks an arbitrary set $S \in \text{ANDset}(root)$

¹In chapters 5 and 6 "with high probability" (w.h.p.), means "with probability at least $1 - O(n^{-\alpha})$ for some constant $\alpha > 0$, for a system with n processors.

and probe its members. If S turns out to be alive then the algorithm may finish. Otherwise, the set is being ‘corrected’, by replacing the choices that turned out to be *dead*.

Let $\text{parent}(v)$ denote the internal node in the tree which is the parent of v . To correct a choice of a *dead* processor $v \in S$, the leaves in the subtree $v' = \text{parent}(v)$ are probed. If the newly probed processors contain a live member of $\text{ANDset}(v')$ then the algorithm can finish. Otherwise, the last step is recursively repeated with $\text{parent}(v')$. This process runs in parallel for each faulty processor $v \in S$. Following from Observation 5.8, when all processes stop, the probed elements contain a live member of $\text{ANDset}(r)$. If a live member exists, then by the time all the process reach the root of the tree, this live member is found. Algorithm 5.1 describes this process more formally.

Algorithm - Find Live Quorum

Input: A configuration $\mathbf{x} \in \{0, 1\}^n$ of the processors).

Output: A live member of $\text{ANDset}(r)$ (or ‘fail’ in case none was found).

1. Probe the members of some set $S \in \text{ANDset}(\text{root})$.
2. For each $v \in S$ which is *dead*, do the following:
 - (a) Set $v' = \text{parent}(v)$
 - (b) Probe all leaves of v'
 - (c) While the leaves in the subtree v' do not contain a live member of $\text{ANDset}(v')$, set $v' \leftarrow \text{parent}(v')$ and go back to stage 2b.
 - (d) Add the members of the live set to S .

Table 5.1: Algorithm - Find a Live Quorum.

We turn to the analysis of the probe complexity of the adaptive algorithm. Step 2 of Algorithm 5.1 finds for each $v \in S$ its lowest ancestor v' , for which a live member in $\text{ANDset}(v')$ exists. Note that if v is alive then $v' = v$. Denote by h_v the distance from v to this v' and by X_v the number of leaves in the subtree v' . h_v determines the number of rounds for each process. The number of rounds of the algorithm is $\max_{v \in S} \{h_v\}$. Lemma 5.10 states that the expected height h_v is constant and that w.h.p. for all $v \in S$, $h_v \leq 2 \log \log n$.

Lemma 5.10. *Let $v \in S$, then $E[h_v] = O(1)$, where the expectation is over the possible*

configurations. Moreover,

$$\Pr\left[\bigwedge_{v \in S} h_v \leq 2 \log \log n\right] = 1 - O\left(\frac{1}{\sqrt{n}}\right).$$

Proof. Step 2c reaches height $h + 1$, only if there was no live member of $\text{ANDset}(v')$ when v' is of height h . As Lemma 5.5 states, the probability for this event is less than $2^{-2^{\frac{h}{2}}}$ for small enough failure probability p . Thus, the expected height reached is:

$$\mathbb{E}[h_v] \leq \sum_{h=0}^{\infty} 2^{-2^{\frac{h}{2}}} = O(1).$$

The second part of the theorem follows directly from the analysis of the non-adaptive algorithm (proof of Theorem 5.9). \square

When $h_v = O(1)$ then the number of leaves probed, $X_v = 2^{h(v)} = O(1)$. The probe complexity of the adaptive algorithm is less than or equal the sum $\sum_{v \in S} X_v$ (its not merely the sum since two trees may intersect). The expected probe complexity is then $O(\sqrt{n})$ (from linearity of expectation). The random variables $\{X_v\}_{v \in S}$ are not independent though. For this reason Chernoff inequality cannot be used directly to bound the probability of a large deviation of this sum. A different argument is now used.

Given a configuration \mathbf{x} of the processors and a set $S \in \text{ANDset}(r)$ (chosen in step 1 of the algorithm), let $g_S(\mathbf{x})$ indicate whether all heights (h_v) are at most $2 \log \log n$, i.e.,

$$g_S(\mathbf{x}) = \begin{cases} 1, & \forall v \in S, h_v \leq 2 \log \log n, \\ 0, & \text{otherwise} \end{cases}$$

Let \mathcal{E} denote the event that all the heights are bounded, namely

$$\mathcal{E} = \{\mathbf{x} \in \{0, 1\}^n \mid g(\mathbf{x}) = 1\}.$$

Lemma 5.10 states that \mathcal{E} is likely to occur, namely,

$$\Pr[\mathcal{E}] = 1 - O\left(\frac{1}{\sqrt{n}}\right).$$

To bound the probability of a large deviation we use the fact that when \mathcal{E} occurs, the variables X_v have a very limited dependency. The set $\{X_v\}_{v \in S}$ can then be divided into $\log n$ disjoint sets of size $\theta(\sqrt{n}/\log n)$ each containing only mutually independent variables. In the event \mathcal{E} , each v' has an ancestor in distance $2 \log \log n$ from the leaves. Denote this ancestor by $\text{ancs}(v')$. Random variable X_v, X_u are independent iff $\text{ancs}(v) \neq \text{ancs}(u)$. Let us

color the leaves in S with $\theta(\log n)$ colors, such that leaves with the same ancestor are colored differently. Denote with A_i the set of members of S with color i . Because leaves colored the same have different ancestors, the members of $\{X_v\}_{v \in A_i}$ are stochastically independent. Assigning the details into the appropriate Chernoff-Hoeffding inequality [5] bounds we get Proposition 5.11.

Proposition 5.11. *Let A_i be a subset of S as described above then,*

$$\Pr\left[\left|\sum_{v \in A_i} X_v - \sum_{v \in A_i} \mathbb{E}[X_v]\right| > \delta |A_i| \mid \mathcal{E}\right] < 2 \exp\left(-\frac{\delta^2 |A_i|}{\log^2 n}\right) = \exp\left(O\left(-\frac{\sqrt{n}}{\log^3 n}\right)\right).$$

Taking a union bound on the $\log n$ events $\{|A_i| = \theta(\frac{\sqrt{n}}{\log n})\}_{1 \leq i \leq \log n}$, we get that

$$\Pr\left[\left|\sum_{v \in S} X_v - \sum_{v \in S} \mathbb{E}[X_v]\right| > \delta |\mathcal{E}| \mid \mathcal{E}\right] < \log n \cdot \exp\left(O\left(-\frac{\sqrt{n}}{\log^3 n}\right)\right) = \exp\left(O\left(n^{-0.49}\right)\right).$$

Since \mathcal{E} occurs w.h.p., a simple union bound argument can be used to show that the probe complexity of Algorithm 5.1 is $\theta(\sqrt{n})$ w.h.p.

Theorem 5.12. *The number of elements probed by the adaptive algorithm is w.h.p. $O(\sqrt{n})$. The number of rounds required in Step 2c of the algorithm is w.h.p. at most $2 \log \log n$.*

Early Stopping

We showed that a live quorum can be found in the And-Or quorum system when each processor fails with some constant probability p , in time $\theta(\log \log n)$ w.h.p. It is desirable that in a different fault model, for example when the failure probability of a processor is $o(1)$, the running time will shorten, namely, achieving early stopping of the algorithm. When no faults occur at all, the adaptive algorithm finds a live quorum in a single round. When the failure probability is $O(\frac{1}{nc})$ for some constant $c > 0$, then w.h.p. no subtree generated in the adaptive algorithm grows to a height of more than a constant. It can be shown in this case that w.h.p. the number of steps required from the algorithm is a constant.

5.3 The And-Or Quorum System on a Balanced Binary Tree

Till now we have discussed the And-Or quorum system in the case the And-Or structure is defined over a complete binary tree. In Chapter 6 we present two ways of maintaining

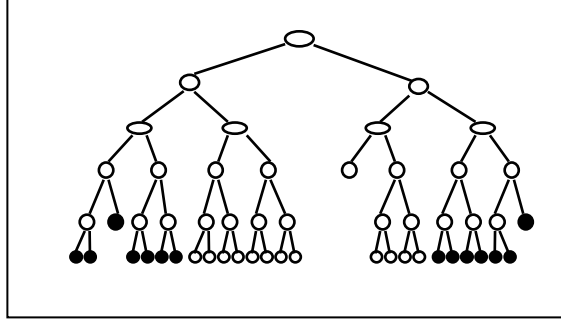


Figure 5.2: A Balanced Tree with max difference of 2 between the highest and lowest leaves. The black leaves form a member of $\text{ANDset}(\text{root})$.

a binary tree over a dynamic environment of processors which is incomplete. In such a case the definition of the recursive structures, $\text{ANDset}(\text{root})$ and $\text{ORset}(\text{root})$ remains well defined. Lemma 3.3 can be extended to show that the intersection property is maintained, so the $\text{AndOr}(\text{root})$ structure is a quorum system. However, the load, availability and probe complexity of the new system are no longer guaranteed to be optimal as in the original And-Or system.

A family of trees is a set of trees $\{T_i\}_{i=1}^{\infty}$ such that T_i has i leaves.

Definition 5.13 (Balanced Tree). A family $\{T_i\}_{i=1}^{\infty}$ of trees is said to be balanced if the following holds:

- Each node in the tree is either a leaf or has two children.
- There exists a constant $c \geq 0$ s.t. for each tree T_i , the difference between the levels of the leaves is bounded by c .

Figure 5.2 demonstrates the And-Or structure on a balanced binary tree.

When the And-Or quorum system is defined over a *balanced* family of trees, the good asymptotic properties are maintained. We will show now, that the load on such a system remains $O(\frac{1}{\sqrt{n}})$, the availability is high and that the algorithms of Section 5.2 keep their optimal probe complexity.

Theorem 5.14. Let T be a tree in a balanced family of trees with n leaves. Let $\mathcal{Q} = \text{AndOr}(T)$ be the And-Or quorum system defined on T . The load of \mathcal{Q} is $\mathcal{L}(\mathcal{Q}) = O(\frac{1}{\sqrt{n}})$.

Proof. It suffices to present a strategy with a load of $O(\frac{1}{\sqrt{n}})$. Let h be the highest level of the tree which is complete, i.e. there are 2^h internal nodes at level h . The tree is balanced,

so it is guaranteed that

$$\lceil \log n \rceil - c \leq h \leq \lceil \log n \rceil \quad (5.3)$$

for some constant c that is independent of n .

Consider the following algorithm for selecting a set $S \in \text{ANDset}(\text{root})$:

1. Pick a set $V \in_R \text{ANDset}(r, h)$ uniformly at random over all the sets in $\text{ANDset}(r, h)$.
2. For each node $v \in V$ uniformly pick a set $V_v \in_R \text{ANDset}(v)$.
3. return $S \triangleq \bigcup_{v \in V} V_v$.

Following Observation 5.8, S returned in step 3 of the algorithm is a member of $\text{ANDset}(\text{root})$. The probability of a node in level h to be chosen into the set V , in step 1 is $\theta(\frac{1}{\sqrt{2^h}}) = \theta(\frac{1}{\sqrt{n}})$ (by Equation 5.3). This follows from the optimality of the uniform strategy [22]. A processor is chosen only if it is a descendant of a node $v \in V$. The probability of a processor to be chosen is then $\theta(\frac{1}{\sqrt{n}})$.

Similarly, a strategy with load $\theta(\frac{1}{\sqrt{n}})$, for choosing a member of $\text{ORset}(\text{root})$ exists. A strategy for choosing a quorum in the balanced And-Or quorum system, with this load then follows. \square

A balanced And-Or tree quorum system also keeps the high availability property of the original And-Or system. This is guaranteed when the failure probability p of a single processor is smaller than some constant threshold $p_c > 0$ that depends only on the difference between the lowest and highest leaves in the balanced family of trees.

Theorem 5.15. *Let T be a tree of n leaves, in a balanced family then,*

$$F_p(\text{AndOr}(T)) \leq \exp(-O(\sqrt{n})),$$

when $p \leq p_c$.

Proof. It suffices to show that the $\text{ANDset}(\text{root}(T))$ set system has the above availability. The availability of the $\text{ORset}(\text{root}(T))$ set system can be shown in a similar way. Consider a binary tree B , not necessarily complete, with a height smaller than some constant c . The failure probability of B depends only on c and p . There are finite number of trees of height at most c , so the global fail probability of the $\text{ANDset}(\text{root}(B))$ set system is upper bounded by a constant that depends on c and p . Denote this bound with $\psi(c, p)$, i.e.,

$$F_p(\text{ANDset}(B)) \leq \psi(c, p) \quad (5.4)$$

Let h be the highest level in T which is complete. Consider a node v in level h . Notice that v is a subtree in T with at most c levels. We say v is *alive*, if and only if there exists a live member of $\text{ANDset}(v)$. Following Equation 5.4, v is alive with probability at least $1 - \psi(c, p)$. Following Observation 5.8 a live member of $\text{ANDset}(\text{root}(T))$ exists, iff there exists a live member in $\text{ANDset}(r, h)$.

It follows from Lemma 5.5 that when a node v in level h fails with probability at most $\frac{1}{4}$, then the global failure probability of T is $F_p(\text{ANDset}(T)) \leq \exp(-O(\sqrt{n}))$. For small enough p ($p \leq p_c$), $\psi(p, c)$ can be made smaller than $\frac{1}{4}$. Hence, $F_p(\text{ANDset}(T)) = \exp(-O(\sqrt{n}))$. \square

Chapter 6

The Dynamic And-Or Quorum System

In this chapter we present an adaptation of the Balanced And-Or quorum system over a dynamic network. We present two implementations: The first is on top of the Identity-Management scheme [17]. In this scheme a balanced binary tree structure is maintained over the active processors in the network and the embedding of the quorum system is straightforward. The second is on top of the distance-halving network. The network communication lines correspond to edges of the binary tree. This implementations saves in routing messages.

For the entire chapter, we make the same simplifying assumption that was used in Chapter 4: An underlying layer takes care of concurrent operations of Join/Depart/‘Use Quorum’, so that the network implementation gets these operations sequentially (where ‘Use Quorum’ is an operation that sends a message to all the members of a quorum).

6.1 A dynamic And-Or Quorum System Using the Identity-Management Scheme

The implementation of the And-Or quorum system in a dynamic environment uses the Identity-Management scheme presented by Manku [17]. In this scheme a balanced binary tree, with the active processors acting as its leaves, is maintained at all times, as processors join and depart from the system. The binary tree structure defines the hosts’ identities. Only leaf nodes of the tree correspond to a processor’s identity. The internal nodes of the tree are conceptual. The sequence of 0s and 1s along the path from the root to a leaf node constitutes the identity of that leaf. The Identity-Management scheme keeps the tree structure balanced

such that at each moment, all leaves are at one of three bottom levels of the tree, (denoted $B, B \pm 1$). This balanced tree is used as a balanced And-Or tree, which was shown in Section 5.3 to generate a quorum system with optimal properties. Two problems arise though:

Integrity: When processors join or depart the system the quorum sets should be modified.

The integrity of the system is preserved in two aspects: First, The intersection property must hold. Second, the adaptation should keep the properties of the quorum system e.g., load and availability.

Locality: Even if the set of quorums is well defined, it is required of a user to actually know the tree structure in order to choose a quorum. In a dynamic system a global view of the system is hard to maintain and is therefore not assumed. The users of the quorum system are the processors themselves. When a processor selects a quorum it has no knowledge of the complete tree of identities and only has access to its own identity.

To maintain *integrity* when the tree structure changes, all the quorums in use that are no longer legitimate quorums, must be modified. In the Identity-Management scheme there are two distributed algorithms that change the tree structure, one for joining a processor and one for processor departure. When a processor p joins the system, another processor q with identity x , reassigns $x0$ as its new identity and p assigns $x1$ as its identity. When a processor p with identity y leaves the system, a pair of processors q and r with identities $x0$ and $x1$ respectively, make the following changes: One of them reassigns y as its new identity, and the other reassigns x as its new identity. These are the only changes to processors' identities that occur in the process.

To maintain a quorum consistent with the tree structure, when join and depart operations occur, it is necessary to perform some action on the quorums in use. A processor p maintains a local variable $p.flag$ that states whether or not it is a member of a quorum in use. The following actions are required to this distributed data structure (see Figure 6.1):

Join: Let q be the processor whose identity is split when processor p joins . Then if $q.flag = false$ then $p.flag := false$. Else, depending on the level (odd/even) in which q is located, either both $p.flag$ and $q.flag$ are assigned true, or randomly, one is assigned true and the other false.

Depart: Let q and r be the pair of processors involved in the departure of processor p . Then set $q.flag := p.flag$ and $r.flag := q.flag \vee r.flag$.

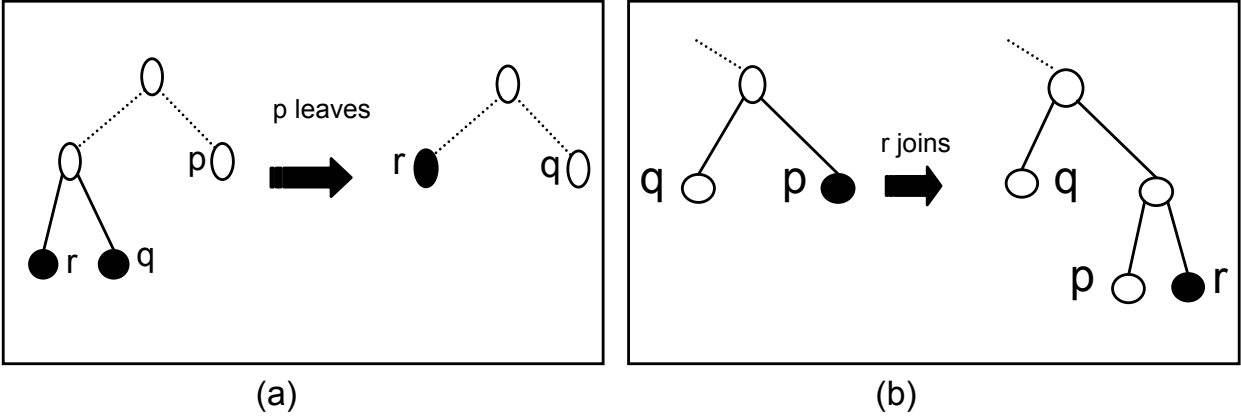


Figure 6.1: (a) p departs from the network. (b) r joins the network. p was a member of a quorum in use, and acted as an OR gate.

Notice that these operations guarantee that a used quorum is modified so that it is always a member of the current And-Or system. Therefore, the good properties of the balanced And-Or system are maintained.

To address the issue of locality we show how a processor can send messages to a random quorum without any knowledge of the tree structure. We use the fact that a processor is a leaf in one of three bottom levels. A processor can deduce its level from its own identity (if the identity is x , then the level is $|x|$). Notice that for each processor identity x , level $|x| - 2$ is guaranteed to be complete. We show now how to select a random member of $\text{ANDset}(\text{root})$. A member of $\text{ORset}(\text{root})$ can be similarly chosen.

1. A set Q is randomly chosen from the members of $\in \text{ANDset}(\text{root}, |x| - 2)$.
2. For each identity $y \in Q$, a message is sent to all elements in a random member of $\text{ANDset}'(y)$ on a complete tree with 5 levels.

Recall the ANDset' operator defined in Section 3.3.3 which includes all nodes visited by the recursive procedure that generates a set in the ANDset collection. Some of the destinations may not represent processors, for which case we assume that messages sent to them simply get lost.

A processor identity y is guaranteed to be between levels $|x| - 2$ and $|x| + 2$, i.e., included in the 5 levels below $|x| - 2$. Thus, the message is sent to a set of processors that includes a quorum on the global tree structure. It is clear that the total number of messages sent is within a multiplicative factor of a quorum size.

6.2 A Dynamic And-Or Quorum System Over the Distance Halving Network

In this section we show an adaptation of the balanced And-Or quorum system of Section 5.3 to a dynamic environment using the distance halving (DH) network of Naor and Wieder [19], that was presented in Section 4.1. This implementation in contrast to the one of Section 6.1, strongly couples the quorum system with the routing scheme of an actual network, saving in communication complexity of algorithms for selecting a quorum.

The embedding of a binary tree over the DH-network, that was used in Section 4.1 is used here to define a dynamic And-Or quorum system. For a set $S \subset I$ and an identities vector \vec{x} , we denote by $\text{CP}_{\vec{x}}(S)$ the set of processors that cover the points in S , i.e., $\text{CP}_{\vec{x}}(S) \triangleq \{p_i \mid \exists s \in S, s \in s(x_i)\}$. In case \vec{x} is clear from context we omit it and simply write $\text{CP}(S)$.

An important parameter of the DH-network studied in [19] is the ratio between the largest and smallest cells induced by a composition. More formally:

Definition 6.1 (From [19]). *The smoothness of \vec{x} denoted by $\rho(x)$ is defined to be $\max_{i,j} \frac{|x_i|}{|x_j|}$. If $\rho(x)$ is a constant independent of n we say that \vec{x} is smooth.*

The smoothness of a network is maintained by the join/depart algorithms, in which processors choose their identities. Several methods for achieving smoothness are described in [8, 13, 17, 1, 19]. The Identity Management scheme [17] achieves a smoothness factor $\rho = 4$. In this scheme a node can also approximate the value of $\log n$ up to an additive constant of ± 2 . In the following implementation of the And-Or quorum system we assume both the network is kept smooth at all times, with a constant factor ρ , and that each processor has an approximation of $\log n$ as mentioned.

6.2.1 The Dynamic And-Or Quorum System Based on a Smooth Distance Halving Network

We first assume that at any moment, each processor has a local variable $l = \lceil \log n \rceil$, where n is the number of currently active processors in the network. We will later show the modifications required in case each processor holds only an approximation of l within an additive constant.

The set of quorums is defined using an embedding of a binary tree T in the DH-graph, rooted in an arbitrary fixed point $r \in I$ with depth l . A set $S \in \text{ANDset}(r)$ is a set of points

in I and so are sets in $\text{ORset}(r)$. T has between n and $2n$ leaves. The set of quorums is defined as the set of processors that cover a quorum in $\text{AndOr}(T)$, i.e.,

$$\mathcal{Q} = \{S \cup R \mid S = \text{CP}(S'), R = \text{CP}(R'), S' \in \text{ANDset}(r), R' \in \text{ORset}(r)\}.$$

Observation 6.2. *For any identities vector assignment \vec{x} , \mathcal{Q} is a quorum system.*

Theorem 6.3. *Let \mathcal{Q} be the dynamic And-Or quorum system defined as above. If \vec{x} is smooth then the load $\mathcal{L}(\mathcal{Q}) = O(\frac{1}{\sqrt{n}})$.*

Proof. Let \mathcal{S} be the quorum system $\mathcal{S} = \text{AndOr}(r)$ defined on a subset of points in I and let w be an optimal strategy for \mathcal{S} with load $O(\frac{1}{\sqrt{n}})$. The quorum $\text{CP}(S) \in \mathcal{Q}$ is selected where S is chosen in accordance with w . In a smooth network, each processor covers at most a constant number of leaves, so the theorem follows. \square

6.2.2 The Adaptation of a Quorum to Network Updates.

In the construction suggested in Section 6.1 each processor was identified with a single leaf in the And-Or quorum system. In the DH based construction, a processor may be responsible for a number of leaves. To keep a quorum consistent with the changing tree structure, it is necessary to perform some action on the quorums in use. Each processor p , maintains a local variable $S(p)$ which is the set of all leaves (points in I), which are members of a quorum in use, and are covered by p . p updates $S(p)$ whenever one of the following events occur: First, when l , the local approximation of the height of the And-Or tree changes. Second, when the segment covered by p shrinks/grows, as a result of neighbor join/depart.

When l increases, each leaf x in the tree splits into $\ell(x), r(x)$. When l decreases, x becomes $\text{parent}(x)$. p updates $S(p)$, by ‘passing’ all members $x \in S(p)$ to either $\ell(x)$ or $r(x)$ or $\text{parent}(x)$ in an obvious way (that also depends on whether it is in an odd or even level in the And-Or tree). In the case where the segment covered by p changes, p either inserts new leaves into $S(p)$ or deletes leaves in $S(p)$ which are no longer covered by it, and passes them to the new processors that cover them. Note that each update to this distributed data structure, can be done by sending a messages via a single edge in the DH-network.

When l is not the exact value of $\lceil \log n \rceil$, but only an approximation within a constant factor, a similar solution to the one in Section 6.1 is used.

6.2.3 Analyzing the Availability and Probe Complexity of the Dynamic And-Or.

In the dynamic And-Or environment we say a point $x \in I$ is alive if and only if the processor that covers it is alive. Otherwise it is said to be dead. A subset of points in I is said to be alive iff all its members are alive. Note that a live quorum of processors in the dynamic And-Or exists, iff a live quorum (of points) exists on the embedded And-Or structure.

In the And-Or system the analysis of F_p was kept relatively easy due to the fact that the failure probabilities of two disjoint subtrees are independent. In the dynamic case however, we cannot compute F_p in the same way: though processors fail independently, the failure of leaves in the tree is not independent. The failure events of two leaves are dependent if and only if they are covered by the same processor. In a smooth network each processor covers at most a constant number of leaves in the embedded tree, so the event that a leaf fails depends on at most a constant number of other such events.

Due to this limited dependency, we can use the technique of domination by product measure which was introduced by Liggett et al.[11], and was used by Naor and Wieder in [20] in a similar context. The intuition behind this technique is that the above configuration is very similar to a configuration where each processor covers exactly one leaf, in which case leaves fail independently of each other and the availability analysis is as in the classic network.

Domination by Product Measure

We Define ‘stochastic domination’ in our context. Say we have a finite set S and a state space $\Omega \triangleq \{0, 1\}^S$. The set S in our case is the set of leaves x_1, \dots, x_m in the embedded tree and Ω the set of all possible configuration (a configuration states which leaves are covered by live processors and which by dead ones). Given $\omega_1, \omega_2 \in \Omega$ we say $\omega_1 \leq \omega_2$ if $\forall s \in S, \omega_1(s) \leq \omega_2(s)$. In our case, $\omega_1(s) \leq \omega_2(s)$ if all surviving leaves in ω_1 also survived in ω_2 .

Given a function $f : \Omega \rightarrow \mathbb{R}$ we say that f is increasing if

$$\omega_1 \leq \omega_2 \Rightarrow f(\omega_1) \leq f(\omega_2).$$

In our case for instance, the function that assigns the value 1 to a configuration that contains a live quorum is an increasing function.

Given two probability measures μ, ν on Ω , μ stochastically dominates ν if for any increasing function f we have $E_\mu(f) \geq E_\nu(f)$. We denote this by $\mu \preceq \nu$. For example, denote by

π_p the product probability measure in which each leaf independently fails with probability p . It is intuitive (though requires proof) that $\pi_{p_1} \preceq \pi_{p_2}$ whenever $p_1 \geq p_2$.

For the availability analysis we fix \vec{x} , the vector of identities in a smooth network. The vector \vec{x} induces the dynamic quorum system on the processors. Denote by m the number of leaves in the And-Or tree. Denote by μ the probability measure induced on the configurations ω of the leaves x_1, \dots, x_m , when each processor independently fails with probability p . We use the following facts in the availability analysis:

1. $\Pr_\mu[\omega(x_i) = 1] = \Pr[\text{CP}(x_i) = 1] = p$. This fact clearly follows from the definition of CP.
2. $\forall i, \omega(x_i)$ stochastically depends on at most c other variables $\omega(x_j)$, where c is a constant independent of m . This fact follows from the smoothness of the network: The number of leaves $m = \theta(n)$ (since there are $\lceil \log n \rceil$ levels plus a constant factor). The two events, that leaf x_i failed and leaf x_j failed are dependent, only if x_i and x_j are covered by the same processor. When the network is smooth, no processor covers more than a constant number of leaves.

It turns out, from a result by Liggett et al [11], that these two conditions suffice for the existence of a product measure $\pi_{p'}$ on $\{0, 1\}^S$ (for some constant $0 < p' \leq p$), which is dominated by μ . Theorem 1.3 in [11] could be stated in our case as follows:

Theorem 6.4 (Liggett et al.). *Let μ be some probability measure on the set of configurations of the leaves x_1, \dots, x_m . Assume that each leaf x_i fails with probability at most p and that the state of each leaf is dependent on the state of at most k other leaves, for some constant k . Then there exists some p' which is a function of p and k , and independent of m such that $\pi_{p'} \preceq \mu$. Furthermore, by decreasing p , p' could be made arbitrarily close to 0.*

Thus $E_\mu(f) \geq E_{\pi_{p'}}(f)$, where f indicates whether a live quorum exists on the leaves. This means that the probability a live quorum exists in μ is larger(or equal) than in $\pi_{p'}$. Hence,

$$F_p = \Pr_\mu[f(\omega) = 0] \leq \Pr_{\pi_{p'}}[f(\omega) = 0].$$

For p small enough, p' can be made smaller than $\frac{1}{4}$, in which case Theorem 5.15 can be used to bound the failure probability:

$$F_p \leq \Pr_{\pi_{p'}}[f(\omega) = 0] \leq \exp(-\theta(\sqrt{n})). \tag{6.1}$$

The following theorem concludes this discussion:

Theorem 6.5. *There exists a constant $p_c > 0$ such that for every $p \leq p_c$, the failure probability of the dynamic And-Or quorum system over a smooth DH-network is $F_p = \exp(-\theta(\sqrt{n}))$.*

Probe Complexity of the Dynamic And-Or

We use essentially the same algorithms described in Section 5.2, but here we first select a set of leaves and then use the set of processors that cover them. The analysis here is more complicated. The events that leaves in different subtrees contain live sets are no longer independent since two points in disjoint trees can be covered by the same processor.

First we show that the non-adaptive algorithm applied to the dynamic case finds a live quorum w.h.p., and therefore the dynamic And-Or retains $O(\sqrt{n} \log n)$ probe complexity. The proof of Theorem 5.9 used the fact that for $p \leq \frac{1}{4}$, an And-Or quorum system on a subtree of height $2 \log \log n$ has a failure probability $F_p \leq \frac{1}{n}$. Theorem 6.5 for the dynamic And-Or stated that for p small enough, the same applies.

Theorem 6.6. *There exists a constant $p_c > 0$ s.t., when the failure probability $p \leq p_c$, then the set of processors chosen by the non-adaptive algorithm (Section 5.2), contains a live quorum w.h.p.*

Proof. Denote by R , the set of points chosen from the collection $\text{ANDset}(\text{root}, \lfloor \log n - 2 \log \log n \rfloor)$. Let $m = |R|$ ($m = O(\frac{\sqrt{n}}{\log n})$). Denote by E_i the event that a subtree of height $2 \log \log n$ defined on a point in R has a live quorum. Note that any $k+1$ events $E_i, E_{i_1}, \dots, E_{i_k}$ which are not independent are positively correlated, i.e.,

$$\Pr[E_i | E_{i_1}, \dots, E_{i_k}] \geq \Pr[E_i],$$

because given that one subtree has a live quorum, can only raise the probability another subtree has a live quorum. The probability that the set S contains a live quorum is then:

$$\Pr[S \text{ contains a live quorum}] = \Pr\left[\bigwedge_{i=1}^m E_i\right] \geq \prod_{i=1}^m \Pr[E_{i_1}] \geq \left(1 - \frac{1}{n}\right)^{\sqrt{n}} = 1 - O\left(\frac{1}{\sqrt{n}}\right).$$

This concludes the proof of Theorem 6.6. □

The number of processors chosen is upper bounded by the number of leaves chosen. Thus, the probe complexity remains $O(\sqrt{n} \log n)$.

The adaptive algorithm for the dynamic case also works in the same way as in the classic case. Again, Theorem 6.5 can be used to show that the local subtrees generated in step 2c

of Algorithm 5.1, have an expected constant height. Hence, on average the algorithm probes $\theta(\sqrt{n})$ processors. However, unlike in the classic case, here there is dependency between the height of different subtrees, so Chernoff bound cannot be used to bound the probability of large deviation. Following the Markov inequality, the probability for $\omega(\sqrt{n})$ probes cannot be a constant. Hence, with probability $1 - o(1)$, the number of probes required is $\theta(\sqrt{n})$. However, we did not show this happens with probability $1 - n^{-\alpha}$, $\alpha > 0$. Therefore, bounding the probe complexity of the adaptive algorithm with a bound lower than $O(\sqrt{n} \log n)$ remains an open problem.

A Gossip Based Implementation

The fact that the edges of the And-Or tree correspond to real communication lines of the DH-network enables a gossip based protocol which saves in the message complexity of both the adaptive and non-adaptive probe algorithms.

The dynamic And-Or presented in Section 6.1, requires that a message be sent to each of the processors selected. Let r denote the expected number of routing hops per message. r is typically logarithmic in the number of processors [1, 19, 29, 31]. If a message is sent to m processors, the number of messages actually sent is blown to rm . In the And-Or system based on the DH-network, messages can percolate through the nodes that cover the tree. Next we show how a message is economically sent to a randomly chosen quorum. When we say a message is sent to point $x \in I$ we mean it is sent to the processor that covers x .

When a processor wishes to send a message m to a quorum of processors it simply writes m to $root = \frac{1}{2}$ (where $\frac{1}{2}$ was chosen arbitrarily from $[0, 1)$). Each processor, upon receiving the message (and its destination x) does the following, according to its *level*.

1. If *level* = $\log n$, accept the message.
2. else, if the *level* is odd, uniformly choose $d \in \{r, \ell\}$ and send m to $d(x)$
3. else, *level* is even. Send m to both $\ell(x)$ and $r(x)$.

The level is the number of steps from the root and can be easily deduced from x . The total number of messages sent in this way can be shown to be $c \cdot \sqrt{n}$ where c is a small constant. The same procedure requires $O(r\sqrt{n})$ messages in the And-Or system of Section 6.1.

Notice that use of a gossip protocol eliminates the need for the DH-routing protocol, during a probing algorithm.

	Dynamic And-Or	Dynamic Paths[20]	Dynamic PQS[2]
Load	$\theta(\frac{1}{\sqrt{n}})$	$\theta(\frac{1}{\sqrt{n}})$	$\theta(\frac{\sqrt{\log n}}{\sqrt{n}})$
Failure Probability	$e^{-\Omega(\sqrt{n})}$	$e^{-\Omega(\sqrt{n})}$	$e^{-\Omega(n)}$
Adaptive Algorithmic PC	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n} \log n)$
Number of rounds	$\theta(\log \log n)$	$\theta(\sqrt{n})$	1
Critical p	$0 < p_c \leq \frac{1}{4}$	$\frac{1}{2}$	-

Table 6.1: A comparison between the dynamic And-Or, the dynamic Paths and the dynamic probabilistic quorum system.

6.3 Comparing with Other Dynamic Quorum Constructions.

Comparing with other dynamic constructions of quorum systems, the *dynamic And-Or* system holds some advantages. The ‘*probabilistic quorums*’ for dynamic network requires a quorum size of $O(\sqrt{n \log n})$ to guarantee intersection w.h.p., whereas the dynamic And-Or has a quorum size of $O(\sqrt{n})$.

The adaptive algorithm of the ‘*Paths*’ quorums system [22] achieves $O(\sqrt{n})$ probe complexity as was shown by Naor and Wieder in [20]. However, the number of rounds in which their adaptive algorithms operates is $O(\sqrt{n})$. The adaptive algorithm for the And-Or (and for the dynamic And-Or) also achieves a $\theta(\sqrt{n})$ algorithmic probe complexity, but requires only $O(\log \log n)$ rounds. In the *Dynamic Paths* [20], whenever a processor joins or depart from the system a local computation of a Voronoi diagram is required. In the dynamic And-Or, join/depart events only require the insertion/deletion of a constant number of values to a set.

It was shown in [23] that for every quorum system, the global failure probability $F_p \xrightarrow{n \rightarrow \infty} 1$, when $p > \frac{1}{2}$. A ‘critical failure probability’ is the maximum p , for which the system is available i.e., for which F_p converges to 0 as n goes to infinity. It was shown in [20] that the critical probability for ‘*Dynamic Paths*’ is $\frac{1}{2}$. The critical value for the And-Or system is $\alpha = \frac{3-\sqrt{5}}{2} < \frac{1}{2}$. The best failure probability p for which we have shown availability in the Dynamic And-Or, is a constant greater than 0, but strictly less than α . The actual critical value remains unknown, but in any case the ‘*Dynamic Paths*’ achieves a better result in this criteria. A detailed comparison between these dynamic quorum systems is in Table 6.3.

Chapter 7

Conclusions and Open Questions

7.1 Storage Systems in Adversarial Fault Model

In Chapter 2 we considered the non-adaptive reader model. In this model the adversary chooses a set T of processors of her choice and crashes them. We showed lower bounds on the read complexity, write complexity and blowup ratio of a storage system, as a function of $|T|$. In Chapter 3 we considered a hybrid model, where the reader is fully adaptive and the adversary can only act in a limited number of rounds. For this model we presented a storage system with a constant blowup-ratio.

What happens in a model where both the reader and the adversary are fully adaptive? We conjecture that the same lower bounds shown for the non-adaptive case still hold in this model. Namely, the product of the read complexity and write complexity must be at least $\Omega(|T|)$ and the product of the read complexity and the blowup-ratio must be at least $\Omega(|T|)$. We know this is true for limited cases, but we haven't been able to prove this conjecture for the general case.

The storage system presented in Chapter 3 has a $\theta(\sqrt{n})$ write complexity, $\theta(\sqrt{n} \log n)$ read complexity and a constant blowup-ratio. As stated in Section 3.2, we cannot expect the product of the read complexity and the write complexity to be $o(n)$ due to the fact that adaptiveness cannot help the reader before the first element is found. The product of the read complexity and the write complexity in our construction is $\theta(n \log n)$. Closing this gap remains an open problem. A related question is to the number of rounds required of a reader in this model. In our storage system a reader operates in $\log n$ rounds. We ask whether a construction using a constant number of rounds exists.

7.2 Algorithmic Probe Complexity of Quorum Systems

We have shown that the And-Or quorum system has an adaptive algorithm for finding a live quorum, that operates in $\log \log n$ rounds and has a $O(\sqrt{n})$ probe complexity. Is there a quorum system with an optimal load for which there is an adaptive algorithm that works in a constant number of rounds (and the same probe complexity)? Recall that Naor and Wieder showed a lower bound on the probe complexity as a function of the load for non-adaptive algorithms (Theorem 5.4). This can be viewed as a lower bound on single round algorithms. A more general goal would be to extend this lower bound for adaptive algorithms with an arbitrary number of rounds. A milestone would be to prove the optimality of our algorithm for finding a live quorum in the And-Or system, or to find an algorithm with better probe complexity.

In Chapter 6 we showed that most of the good properties of the And-Or quorum system are maintained in the dynamic implementation over the DH-network. We showed that the algorithmic probe complexity of our adaptive algorithm is $O(\sqrt{n})$ on expectation and even with probability $1 - o(1)$. However, showing the probe complexity is $O(\sqrt{n})$ w.h.p. remains open as well.

Bibliography

- [1] Ittai Abraham, Baruch Awerbuch, Yossi Azar, Yair Bartal, Dahlia Malkhi, and Elan Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, page 40, 2003.
- [2] Ittai Abraham and Dahlia Malkhi. Probabilistic quorums for dynamic systems. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC)*, pages 60–74, 2003.
- [3] Alvisi, Malkhi, Pierce, Reiter, and Wright. Dynamic byzantine quorum systems. In *International Conference on Dependable Systems and Networks (IEEE Computer Society) (replacing both IEEE FTCS (after 29th) and IFIP DCCA (after 7th))*, volume 1, 2000.
- [4] Rida A. Bazzi. Planar quorums. In *Distributed Algorithms, 10th International Workshop, WDAG '96*, pages 251–268.
- [5] Oded Goldreich. *Randomized Methods in Computation - Lecture Notes*. <http://www.wisdom.weizmann.ac.il/~oded/rnd.html>, 2001.
- [6] Yehuda Hassin and David Peleg. Average probe complexity in quorum systems. In *20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 180–189, 2001.
- [7] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Second International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 98–107, 2003.
- [8] David R. Karger and Matthias Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 36–43, 2004.

- [9] Katz and Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2000.
- [10] Kerenidis and de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *JCSS: Journal of Computer and System Sciences*, 69, 2004.
- [11] Thomas L. Liggett, Roberto H. Schonmann, and Alan M. Stacey. Domination by product measures. *The Annals of Probability*, 25(1):71–95, January 1997.
- [12] Nancy Lynch, Dahlia Malkhi, and David Ratajczak. Atomic data access in distributed hash tables. *Lecture Notes in Computer Science*, 2429:295–??, 2002.
- [13] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *ACM Conf. on Principles of Distributed Computing (PODC)*, pages 183–192, 2002.
- [14] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC '97)*, pages 569–578, New York, May 1997. Association for Computing Machinery.
- [15] Dahlia Malkhi, Michael Reiter, and Wright Wright. Probabilistic byzantine quorum systems, November 30 1999.
- [16] Dahlia Malkhi, Michael K. Reiter, and Rebecca N. Wright. Probabilistic quorum systems. In *Symposium on Principles of Distributed Computing*, pages 267–273, 1997.
- [17] Gurmeet S. Manku. Balanced binary trees for id management and load balance in distributed hash tables. In *Proc. 23rd ACM Symposium on Principles of Distributed Computing, (PODC)*, pages 197–205, 2004.
- [18] Moni Naor and Ron M. Roth. Optimal file sharing in distributed networks. *SIAM Journal on Computing*, 24(1):158–183, February 1995.
- [19] Moni Naor and Udi Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 50–59, 2003.
- [20] Moni Naor and Udi Wieder. Scalable and dynamic quorum systems. In *ACM Conf. on Principles of Distributed Computing (PODC)*, pages 114–122, 2003.

- [21] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In *Second International Workshop on Peer-to-Peer Systems*, February 2003.
- [22] Moni Naor and Avishai Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, 1998.
- [23] D. Peleg and A. Wool. The availability of quorum systems. Technical Report CS93-17, Weizmann Institute of Science, Faculty of Mathematical Sciences, January 1, 1993.
- [24] David Peleg and Avishai Wool. How to be an efficient snoop, or the probe complexity of quorum systems. *SIAM Journal on Discrete Mathematics*, 15(3):416–433, August 2002.
- [25] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, April 1989.
- [26] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proc ACM SIGCOMM*, pages 161–172, 2001.
- [27] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
- [28] J. Saia, A. Fiat, S. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *First International Workshop on Peer-to-Peer Systems*, MIT Faculty Club, Cambridge, MA, USA, 2002.
- [29] Ian Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
- [30] Madhu Sudan. Algorithmic issues in coding theory. *Lecture Notes in Computer Science*, 1346:184–??, 1997.
- [31] B.Y. Zhao and J. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB CSD 01-1141, University of California at Berkeley, Computer Science Department, 2001.