Fault–Tolerant Storage in a Dynamic Environment ⋆

Uri Nadav and Moni Naor

Dept. of Computer Science and Applied Mathematics,
The Weizmann Institute of Science.
E-mail:{uri.nadav, moni.naor}@weizmann.ac.il

Abstract. We suggest a file storage system for a dynamic environment where servers may join and leave the system. Our construction 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 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.

1 Introduction

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 nonadaptive 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.

⋆ Research supported by a grant from the Israel Science Foundation.

This paper deals with the fault-tolerance of storage systems in an adversarial model. The failing processors are chosen maliciously as opposed to a random fault model, where each processor independently fails with some constant probability. 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 can 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 nonadaptive 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 *censor*, because a censor 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 [6, 11, 15, 17, 18] deal with this problem.

1.1 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.[8], is a tuple $(\mathcal{Q}, \mu)$ 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.

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 $(\mathcal{Q}, \mu)$, 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 Section 2 we show that no storage system can do better than this simple construction, in a nonadaptive model. In Section 3 we present a storage system that outperforms this quorum based solution, in terms of data blowup ratio, in a slightly modified model. This is the reason we define and investigate storage systems rather than just deal with QS.

The *load* on a system of processors induced by a strategy μ, defined in [14], captures the probability of accessing the busiest processor by μ. 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 induced by the 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.

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 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.

1.2 A Simple Example of a Fault-Tolerant Storage System

Malkhi et al. give in [8] 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 nonadaptive 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 [1] 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.

1.3 Related Work

There are many models for the problem of fault-tolerant data networks. The works of [12, 17, 18] yield fault-tolerant storage systems for a dynamic environment, In a random fault model. In a much more powerful adversarial model, 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 made of the all files stored in the system concatenated together. The write complexity of such a scheme which is fault-tolerant to $\Omega(n)$ faults is $\Omega(n)$, from properties of error correcting codes. For this reason this approach is less appropriate in a dynamic environment.

Rabin defines an Information Dispersal Algorithm [16] as a storage scheme of a file over n processors, such that every m of them can recover the file. This scheme seems wasteful in our case since the dispersed information is made of all the files stored in the system. A single file cannot generally be retrieved from fewer than m processors.

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. Katz and Trevisan [5] show that a code which is locally decodable from a constant number of queries, cannot have a constant rate.

In [3] a content addressable network is proposed 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 that wishes to eliminate access to a specific file can do it by crashing $\log n$ processors. In contrast we are concerned with a model in which each file can be recovered w.h.p. after adversarial deletions.

1.4 Results and Paper Organization

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.

Bad news: No fault-tolerant storage system with nonadaptive read strategy can do better than the construction discussed in Section 1.2. In particular, it will be shown 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, nonadaptive storage systems that are fault-tolerant against t nonadaptive deletions and have a blowup-ratio ρ, must have read complexity $\Omega(\frac{t}{\rho})$.

The remainder of this paper is organized in the following way. Section 2 discusses fault-tolerant storage system for nonadaptive retrieval model. In Section 3 we present a fault-tolerant storage system, with an adaptive read strategy that maintains a constant blowup-ratio. Both these sections deal with the storage of a single file in a static network. In Section 4 we discuss an adaptation of the above scheme to a dynamic model. We note that all proofs are omitted from this version due to lack of space [1].

2 Storage Systems with Nonadaptive Retrieval

This section deals with the analysis of storage systems with nonadaptive retrieval schemes. A nonadaptive 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 Nonadaptive Retrieval Model

We begin by giving a formal definition of a storage system with nonadaptive retrieval, and of fault-tolerance in the nonadaptive 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 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 write and read strategies respectively. 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. 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 *read/write complexity* of a system as the expected number of processors accessed during a read/write operation.

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 (Nonadaptive Storage System). *A* k*-storage system is a 4-tuple* $(\mu_w, \mu_r, \mathcal{E}, \mathcal{D})$ *consisting of*

- *A write strategy* μ_w *and a read strategy* μ_r*.*
- *Encoding mapping:* $\mathcal{E}(f, q_w) \mapsto (x_1, \ldots, x_n)$ *, 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 coding rule from a file of size* k *bits and a list of memory allocations into processors' memories. A '$*$' represents no data is stored in a processor.*
- *Decoding (reconstruction) mapping:* $\mathcal{D}(x_1, \ldots, x_n) \mapsto \{0,1\}^k$ *. where* x_i *is either a bit string or one of the symbols* $\{\phi, *\}$*. A '$*$' in the ith parameter represents that no data was written to the ith processor. A 'ϕ' in the ith parameter represents that the ith processor was not queried.*

[1] A full version of this paper that includes all proofs is available at authors' homepages.

Let π denote the following projection function from a list of n parameters $(x_1, \ldots, x_n)$ and a characteristic vector $q \in \{0,1\}^n$ into a list of n items such that the ith item is $(\pi((x_1, \ldots, x_n), q))_i = \begin{cases} x_i, \, q(i) = 1, \\ \phi, \text{ otherwise.} \end{cases}$. The projection of $(x_1, \ldots x_n)$ 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 $\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r))$. The process ends in a recovery of f if $\mathcal{D}(\pi(\mathcal{E}(f, q_w), q_r)) \equiv f$.

Definition 3 (Nonadaptive (ϵ, k)-Storage System). *An (ϵ, k) 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.

Information can be encoded implicitly in the way bits are allocated among processors and not only in the stored bits themselves. The number of saved bits encoded this way can have only relatively small contribution, when the number of processors used for writing/reading is small enough compared to the number of stored bits. We therefore choose to ignore storage systems that make use of such implicit encodings. We regard only storage systems in which the decoding procedure of the retrieval process uses only bits which were explicitly stored. These are systems in which the memory allocation vector (q_w) is distributed independently of the content of the stored file f. In this case Theorem 1 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 4 captures this property of a pair of read and write strategies.

Definition 4 ((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 $Pr[\langle q_w, q_r \rangle > k] > 1 - \epsilon$, where $q_w \sim \mu_w$ and $q_r \sim \mu_r$ are independent random variables.*

Theorem 1. *Let $\mathcal{S} = (\mu_w, \mu_r, \mathcal{E}, \mathcal{D})$ be an (ϵ, k) storage system, where the vector of memory allocations, distributed as μ_w, is independent of the content of a file f. Then the write and read strategies (μ_w, μ_r) must satisfy the $(2\epsilon, k)$ intersection property.*

This condition, however, is insufficient. 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 [10] 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 nonadaptive adversary model.

Definition 5 (Nonadaptive 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.*

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 1. *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*

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

Next we turn to analyze fault-tolerant storage systems. We show that in such systems, where the retrieval scheme is nonadaptive, 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. *Let (μ_r, μ_w) be the write strategy and read strategy of a (t, ϵ, k) fault-tolerant storage system. Then*

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

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

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 [7, 9]. In that case, the intersection property is defined as $Pr\big[|Q_r \cap Q_w| > \tau\big] > 1-\epsilon$, where Q_r, Q_w are distributed as μ_r, μ_w respectively. A similar result to Theorem 2 holds in that case.

Theorem 3. *Let (μ_r, μ_w) be the write and read strategies of a (t, ϵ, k) fault-tolerant storage system with threshold τ. Then $E[|q_r|] \cdot E[|q_w|] \geq (1-\epsilon)\tau t$, where $q_r \sim \mu_r$ and $q_w \sim \mu_w$ are independent of each other.*

2.3 Bounding the Blowup-Ratio

We bound the minimum number of bits used for the storage of a single file in the nonadaptive 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 nonadaptive 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 6 (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 $\rho(\mathcal{S}) \triangleq E_{\mu_w}[||q_w||]/k$, 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 nonadaptive.

Theorem 4. *Let $\mathcal{S}$ be a (t, ϵ, k) storage system with a read strategy μ_r, then:*

$$\rho(S) > (1-\epsilon)\frac{t}{E[|q_r|]} \ , \ \textit{where } q_r \sim \mu_r .$$

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

3 Fault-Tolerant Adaptive Storage System

In this section we consider a different model in which the adversary is (almost) nonadaptive 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 nonadaptive) 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 read/write operation intended on i is applied to $\pi(i)$ (the reader knows π after he queries a single processor). Since the adversary, being nonadaptive, 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 section 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 allow to find a large fraction of the ones used for storage. It follows from Theorem 3 that this is not possible in the nonadaptive retrieval model.

3.1 Small Load Implies Some Resiliency Against a Nonadaptive Adversary

We wish to upper bound the probability that a nonadaptive 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 nonadaptive 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 $O(\frac{1}{\sqrt{n}})$ [14]. 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 5 shows that even after a nonadaptive 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 1 that such a reader has an optimal read complexity, up to a constant.

Theorem 5. *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,*

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

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

Corollary 1. $Pr_\mu\big[|T \cap Q| > \lambda E[|Q|]\big] \quad < \quad \frac{|T|\mathcal{L}}{\lambda E[|Q|]} \ .$

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 nonadaptive 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 nonadaptive 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 [16] is used for this purpose. These pieces of data are then distributed to the ℓ members of a selected write set.

Now, let a nonadaptive adversary crash a δ fraction of the processors in the network, where $0 \leq \delta < \frac{1}{2}$. Corollary 1 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 [8] 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 [14] 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 can be used to show this lower bound.

3.3 Adaptive Storage System Based on the 'AndOr' System

We present a pair of write strategy and adaptive read algorithm, based on the *'AndOr'* quorum system shown in [14] which has an 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 *'AndOr'* 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, $ANDset(v) = ORset(v) = \big\{\{v\}\big\}$.

(ii) $ANDset(v) = \{S \cup R | S \in ORset(v.left) \wedge R \in ORset(v.right)\}$.
(iii) $ORset(v) = ANDset(v.left) \cup ANDset(v.right)$

The AndOr 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 AndOr structure induces the following properties:

Lemma 2. *(from [14]) 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 $|S| = 2^{\lfloor \frac{h+1}{2} \rfloor}$.*

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 $R \in$ORset(*root*), as follows from Lemma 2 . 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 $0, 2, 4, \ldots, \log n - 2$. The algorithm is then recursively applied.

Proposition 1. *The read complexity of the algorithm for identifying all the members of $S \in$ANDset(root), where root is the root of a tree of height h, is $O(\sqrt{n}\log n)$, where $n = 2^h$. The number of adaptive rounds is $\log n$.*

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 nonadaptive adversary, where $0 < \delta < \frac{1}{2}$. Let $S \in$ANDset(*root*) be chosen at random as described earlier. Corollary 1 implies that with a constant probability at least half the elements of S are not 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 ORset(*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 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 6 considers the fault-tolerance of the described storage system in the (fully) nonadaptive 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 6 remains correct under this 'almost nonadaptive' 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.

Dynamically Adjusting the Retrieval Scheme to the Number of Faults. A drawback of the AndOr storage system, is that even in the case when much fewer than $\theta(n)$ faults occur the read complexity can not be reduced. In [2] 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 [2], no explicit knowledge of the number of faults is required.

The following modifications to the AndOr 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, $ANDset'(v) = ORset'(v) = \big\{\{v\}\big\}$.
(ii) $ANDset'(v) = \big\{\{v\} \cup R \cup S \mid R \in ORset'(v.left) \wedge S \in ORset'(v.right)\big\}$.
(iii) $ORset'(v) = \big\{\{v\} \cup S \mid S \in ANDset'(v.left) \cup ANDset'(v.right)\big\}$

Let $S \in \text{ANDset}'(root)$. 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 ith level of the tree. For each i, S_i can be thought of as a set belonging to ANDset($root$) 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 $S_0, \ldots, S_h$ in an increasing order. The members of a set S_i can be identified using the algorithm described in Section 3.3 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 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 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.*

4 Dynamic Implementation of the AndOr Storage System

In this section we adapt the storage system of Section 3 to a dynamic environment using the distance halving (DH) dynamic hash table (DHT) of [11]. We remark that the AndOr system can be implemented on other dynamic hash tables such as Koorde [4] but the analysis is kept much simpler using the continuous approach of the DH graph.

4.1 Some Properties of the Distance Halving Dynamic Network

We recall the construction of the DH-network from [11]. We first define the DH-graph G_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: $\ell(y) \triangleq \frac{y}{2}$, $r(y) \triangleq \frac{y+1}{2}$ where $y \in I$, ℓ abbreviates 'left' and r abbreviates 'right'. The edge set of G_c is then $\{(y, \ell(y)), (y, r(y)) |\ y \in I\}$.

Denote by $\boldsymbol{x}$ a set of n points in I. The point x_i is the id. of the *ith* processor, in a network of n processors. The points of $\boldsymbol{x}$ divide I into n segments. Define the segment of x_i to be $s(x_i) = [x_i, x_{i+1})$ $(i = 1, \ldots, n-1)$ 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 $z \in s(x_j)$.

To implement the AndOr system on a DH-network we define the notion of a 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 [11] 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.

4.2 Dynamic fault-tolerant Scheme

We show an implementation of a dynamic storage system, based on the AndOr system over the DH-network. In [11] 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 AndOr 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 $S \in$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 [11].

Theorem 8. *Assume the DH-network is balanced, then the above storage system is $\theta(n)$-fault tolerant.*

We note that when the id. is uniformly chosen in the join process, the system remains $\theta(n)$-fault tolerant. Uniform choice does not achieve 'balancing', 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.

Storage Process. Storage can be done in a direct way by using the original routing of the DH-network (see [11]). Another implementation uses a *'gossip protocol'*. The encoded pieces of data start percolating from the root of $\tau(f)$ until they reach S. On even layers (AND gates) the data is equally divided into two parts, each are then 'gossiped' to a different child. On odd layers (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 randomly chosen from the collection ANDset($root$).

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 .

Retrieval Process. Retrieval is done as in the AndOr 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 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 [6] as part of the Viceroy network. The same technique can also be used in the DH-network.

4.3 Dynamic Quorum System

The AndOr system implementation over the DH-network, can serve as a quorum system in a dynamic environment. We need to show how quorums adapt to changes in the size of the network. When a processor that covers a node (point in I) of a quorum senses that $\log n$ has grown it gossips the node to its children on the next layer as described in Section 4.2. When $\log n$ decreases the processor simply deletes the node. Similar techniques to adapt quorums were used in [1].

The optimal load and high availability properties of the AndOr system [14] are maintained.

In comparison with other dynamic constructions of QS, the dynamic AndOr system holds some advantages: The quorum size is $O(\sqrt{n})$ in contrast with [1], where a quorum size of $O(\sqrt{nlogn})$ is required to guarantee an intersection with probability $1 - o(1)$. The join and leave algorithms of the dynamic *'Paths'* system [13] involves a local computation of the Voronoi diagram, whereas in the dynamic AndOr system, a simple step of replication/deletion is required for these operations. Last, the ability to gossip the quorum in logarithmic number of steps is another advantage of our dynamic construction over the dynamic paths.

5 Acknowledgements

We would like to thank Ilan Gronau for many useful comments and Guy Rothblum for thoroughly reading this paper. We also thank the anonymous referees for their useful comments.

References

1. I.Abraham and D.Malkhi, Probabilistic quorums for dynamic systems. DISC 2003.
2. L. Alvisi, D. Malkhi, E. Pierce, M. Reiter and R. Wright. Dynamic Byzantine Quorum Systems. Dependable Systems and Networks,(DSN) 2000.
3. A.Fiat and J.Saia. Censorship resistant peer-to-peer content addressable networks. SODA 02.
4. F.Kaashoek, D.R.Karger. Koorde, a simple degree optimal hash table. IPTPS 03.
5. J. Katz and L. Trevisan, On the efficiency of local decoding procedures for error-correcting codes, Proc. STOC 2000.
6. D.Malkhi, M.Naor and D.Ratajczak, Viceroy, a Scalable and dynamic emulation of the buterfly. Proc. PODC 2002.
7. D. Malkhi and M. Reiter. Byzantine Quorum Systems. The Journal of Distributed Computing, 11(4) 1998.
8. D. Malkhi, M. Reiter and R. Wright, Probabilistic quorum systems. PODC 97.
9. D. Malkhi, M. Reiter, A. Wool, and R. Wright. Probabilistic Byzantine Quorum Systems. Proc. PODC 98.
10. M.Naor and R.Roth, Optimal File Sharing in Distributed Networks, SIAM J. Comput. 1995.
11. M.Naor and U.Wieder, Novel architectures for p2p applications: the continuous-discrete approach Proc. SPAA 2003
12. M.Naor and U.Wieder, A Simple fault-tolerant Distributed Hash Table, IPTPS 03.
13. M.Naor and U.Wieder, Scalable and dynamic quorum systems, Proc. PODC 03.
14. M.Naor and A.Wool, The load capacity and availability of quorum systems, SIAM J. on Computing, April 1998.
15. S.Ratnasamy, P.Francis, M.Handley, R.Karp and S.Shenker. A scalable content addressable network. In Proc ACM SIGCOMM 2001.
16. M.O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance, J. ACM, 36 (1989).
17. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, ACM SIGCOMM 2001.
18. B.Y. Zhao and Z. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB CSD 01-1141. 2001.