Internet-scale storage systems under churn - A steady state analysis

Anwitaman Datta, Karl Aberer
{anwitaman.datta, karl.aberer}@epfl.ch
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland

EPFL Technical Report: LSIR-REPORT-2005-002

Abstract

Content storage in a distributed collaborative environment uses redundancy for better resilience and thus provide good availability and durability. In a peer-to-peer environment, where peers continuously leave and rejoin the network, lazy strategies are employed to maintain a minimal redundancy of stored content in the system efficiently. We propose a new randomized lazy maintenance scheme which has significant performance benefits in comparison to the existing lazy maintenance scheme which is based on deterministic procrastination. Existing analysis (of static resilience) fail to capture in detail the system's behavior over time since it ignores the crucial interplay between churn and maintenance operations and the fact that the system actually does not reside in its average state and instead there's a probability of the system being in each possible system state. We introduce a Markov analysis methodology to determine the time evolution of this probability distribution. We show that given a fixed rate of churn and a specific maintenance strategy, the system operates in a corresponding steady-state (dynamic equilibrium). We validate the analytically predicted system behavior with simulations. Understanding the behavior of the system under such a dynamic equilibrium is a fundamental ingredient to precisely evaluate the system's resilience as well as to determine the operational cost - using which we not only show the superior performance of our proposed randomized lazy maintenance scheme, but also expose the fact that the previous deterministic lazy maintenance scheme in fact renders the system rather vulnerable.

1 Introduction

In the recent years there has been an increasing trend to use resources at the edge of the network - typically desktop computers interconnected across the internet provide services and run applications in a peer-to-peer manner, as an alternative to the traditional paradigm of using dedicated infrastructure and centralized coordination and control. Similar services in a relatively more dedicated infrastructure like PlanetLab [17] is also a growing trend. One such extensively studied application is that of collaborative storage systems, where free storage space of individual computers is used in order to realize persistent and highly available data storage [4, 7, 10, 12, 19, 22]. Such collaborative storage systems can be used by a wide range of applications - as backup service for individual users [6], public services like digital library or an internet archive [15] or file systems [7] - to name a few. Moreover, the peer-to-peer paradigm need not necessarily be used only out in the open in the internet. Private enterprises spread geographically over various sites can use the same peer-to-peer paradigm for a cost effective automated back-up service within their own corporate intranets.

Large-scale systems in general, and peer-to-peer systems in particular, are prone to the unreliability of individual participants. Particularly in a peer-to-peer environment, individual peers often leave and rejoin the network for relatively shorter session times over a long period of time (lifetime), before leaving the network permanently. Also, some peers may become temporarily unreachable from other peers because of communication network problems. Irrespective of such autonomous and whimsical behavior of individual participants, for any practical usability, it is necessary to design systems which are stable and reliable, even if the individual participants are unreliable. Such reliability is achieved using redundancy. In order to maintain the redundancy over a long period of time, it is also necessary to continuously compensate for the lost redundancy in presence of continuous membership dynamics (churn). The maintenance operation needs to be sufficiently aggressive in order to provide a minimal redundancy and resilience, but at the same time the maintenance overheads need to be kept low. This needs prudent design of the maintenance scheme.

In this paper we propose and analyze our maintenance algorithm for collaborative storage systems which is both efficient as well as robust so that the storage system can tolerate a wide range of churn as well as rarer but likely correlated failures.

The algorithm we propose is a *randomized lazy maintenance* scheme which has quantitatively superior resilience against both a wide range of churn level as well as rarer but inevitable (in large-scale distributed systems) correlated failures than known maintenance scheme [4]. Our maintenance scheme achieves this resilience at a comparable (and often lower) maintenance overhead. The basic idea is to probe randomly for a minimal number of redundant fragments of the stored content. Depending on the current available redundancy the sample size thus automatically adapts, as does the effort to replace the detected lost redundancy.

We evaluate our maintenance scheme analytically. To that end we study *time-evolution and steady state characteristics* of storage systems under churn by introducing analytical tools used in the study of dynamical systems [11]. We specifically investigate whether, given a specific amount of churn and a chosen maintenance mechanism the system works in a steady-state and if so, how stable is the steady state to additional failures (possibly caused abruptly by correlated failures) and what is the maintenance cost. We validate our results with simulations.

Despite the immense interest in building reliable P2P storage systems, the issue of churn is not properly studied, apart resorting to simulations [22] which still do not capture the interplay between - churn, specific properties of the kind of redundancy used and the specific maintenance operations. OceanStore [12, 23] looks only into resilience against disk failures, and is not designed for a dynamic setting as is more likely in a peer-to-peer setting. Other systems like CFS [7] and Glacier [10] eagerly maintains redundancy - which however does not explore the trade-offs of cost and resilience. TotalRecall [4] proposed a heuristic lazy maintenance scheme to reduce the maintenance cost, and used simulations to demonstrate the tremendous cost savings using their heuristic in comparison to the use of eager repairs. What was overlooked there is that the system was actually rendered rather vulnerable.

The paper is organized as follows: In Section 2 we discern the subtle differences in the kind of redundancy one can have for content storage, and the practical implications of these subtle differences. We describe in Section 3 the specifics of the maintenance strategies that can be employed to maintain the redundancy of the stored content, and introduce our randomized lazy maintenance scheme. In Section 4 we review the various models for analyzing churn and advocate the use of time-evolution analysis - a well established methodology (developed and used in diverse domains including physics and cybernetics [11]) for studying large dynamic systems. In Section 5 we describe the specific model of churn which we study in this paper. In Section 6 we perform the time evolution analysis for our randomized lazy maintenance scheme and that of the existing deterministic lazy maintenance scheme. We present our results in Section 7 where we show how our randomized lazy maintenance scheme outperforms the existing maintenance scheme. We summarize some ongoing and future work in Section 8 before concluding in Section 9.

2 Redundancy mechanisms: Replication, Erasures and Digital Fountains

Redundancy is essential to fault-tolerance. Moreover, in a peer-to-peer setting characterized by churn, unavailability of any individual peer is more the rule than the exception. Irrespective of the behavior of individual peers, collaborative storage systems endeavor to provide reliable - i.e., highly available and persistent, storage.

Storage redundancy is typically realized by either purely replicating objects (e.g., CFS [7]), or using erasure codes (e.g., RAID [16]). Hybrid strategies in order to improve access efficiency using replication while providing persistence in a memory efficient fashion using erasure codes is also a standard practice (e.g., HP AutoRAID [24]), which has more recently been used in a P2P setting in various systems like Oceanstore [12] and TotalRecall [4].

Erasure codes (e.g., Reed-Solomon codes [18]) have the property that any *M out-of N* fragments can be used to decode and reconstruct an object O. At a storage overhead slightly more than N/M (since in practice size of the object $|O|$ is slightly smaller than size of M fragments) erasure codes provide much better redundancy than what may be achieved using the same storage overhead if pure mirroring (replication) is used [5, 23]. There are however performance trade-offs in actively accessing data [20], and hence erasure coded redundancy is practical only for providing persistence to relatively larger objects in a storage efficient manner, both because of the direct overheads of decoding (reconstructing the object) as well as other practical considerations in such collaborative storage systems - for instance, managing the information about all the fragments and accessing them, among others [22].

Moreover, even though in principle replication is a special case of erasure codes: *1 out-of N*, there is a subtle practical difference typically ignored in the existing analyses [3, 23]. Note that for non-trivial erasure codes, any M *(but) distinct* fragments are required. Thus, if a node goes offline and rejoins, and in the meanwhile the missing fragment has been replenished by the system's maintenance operation, this replica of the erasure coded fragment does not enhance the availability of the whole object[1]. In contrast,

[1]It potentially can be exploited to enhance the availability of individual fragments but that would lead to higher implementation complexity as well as operational overheads, the benefits of which may be marginal or even detrimental. Or else the duplicate needs to be garbage collected. Whichever be the case, the issue is relevant for system implementation but can be abstracted out in our steady-state analysis, where we use availability of individual distinct fragments (whichsoever way the specific

in a pure replication based system, obviously the duplicates indeed enhance the availability. In that respect rateless or Digital Fountain (DFs) codes [14, 21] is more like replication. Using DFs lead to generation of random and unique fragments, so that whenever a particular fragment is lost/unavailable, there is neither any need to identify specifically which one fragment is missing, nor is there any risk of having duplicate fragments if and when the missing fragment returns to the system (because of peer rejoins).

The above discussion highlights the subtle differences in redundancy realized by replication, traditional finite rate erasure codes and rateless digital fountains. This has implications on the time-evolution. In this paper, due to space constraints as well as to first bridge the gap in existing literature, we'll restrict our analysis to only traditional finite rate erasure codes. For pure replication or DF based redundancy, the same analytical tool can be reused, however the precise details of the analysis will differ, since unlike the case of finite rate erasure codes, using either of pure replication or DFs implies we can potentially have infinite redundancy, even if that's neither necessary nor practical.

While the precise details of the analysis thus depends on the specific properties of the kind of redundancy used, the qualitative results are not expected to be affected. Which is to say, the randomized lazy maintenance scheme we propose will in all cases be more efficient and resilient than the deterministic lazy maintenance scheme.

3 Maintenance strategies

Since individual participants in a peer-to-peer system can autonomously leave the system - either temporarily or permanently, just storing an object with some redundancy is not sufficient. A minimal redundancy needs to be maintained for that object as long as we'd like it to persist in the system, irrespective of whether the original peers which stored the object (fragments) stay or not. This necessitates a suitable maintenance strategy.

Existing P2P storage systems employ either an eager repair strategy, or a deterministic lazy repair strategy [4], as elaborated below. The existing lazy repair strategy outperforms the eager repair strategy in terms of maintenance cost, however, as we'll subsequently show, this strategy actually is rather vulnerable. We propose a randomized lazy maintenance scheme. For the same redundancy (same M out-of N erasure code) and comparable maintenance cost to the deterministic lazy mechanism, our maintenance strategy provides much better resilience against both regular churn, as well as has better resilience against correlated failures. Next we explain what the existing eager and deterministic lazy maintenance strategies do, and then introduce our randomized lazy maintenance scheme.

Eager repair: In this strategy the storage system periodically probes for availability of each peer, and replaces any (possibly and most likely temporarily) unavailable data. Such a proactive maintenance mechanism means the system always operates in a state where redundancy level remains constant apart from temporary reduction between repair periods. However as expected intuitively, and has also been observed using simulations by others [4], such a maintenance strategy is very expensive.

Deterministic lazy repair (Strategy-A): The life-time of participants in a peer-to-peer network is often much longer than its session times - that is to say peers often leave the system temporarily only to rejoin back. Consequently, it is not necessary to always replace all fragments which appear to be unavailable

implementation deals with duplicate fragments).

(of a stored object) as done in a eager repair strategy, and instead lazier repair strategies can be used - particularly for large objects where periodic repairs is prohibitive. TotalRecall [4] exploits this to propose a lazy repair strategy which we call "*deterministic procrastination*".

In the *Deterministic procrastination* approach all peers storing fragments for an object are probed periodically. Repairs are triggered only when a certain threshold T_a of nodes (and corresponding data) becomes unavailable for that specific object. Thus to say, when an object has no more than $T_a > M$ fragments available in the system, then a repair process for the object is initiated so that at the end of the repair process all N fragments are again available. This maintenance strategy is proposed and simulated for the TotalRecall [4] system but the dynamics has not been analyzed. This strategy allows a significant loss of redundancy before triggering many repairs all at the same time. This is undesirable because by waiting before losing a significant amount of redundancy, the system becomes vulnerable to sudden multiple (correlated) failures.

Randomized lazy repair (Strategy-B): Another possible strategy, which we introduce in this paper and call *sampling random subsets*, is to probe only a fraction of the stored fragments randomly (uniformly), until a minimal $T_b \geq M$ number of live fragments are detected. Thus a random number $T_b + X$ of probes (determined according to a probability distribution which depends on the actual number of live fragments) will be required to locate T_b live fragments. Then X fragments which were detected to be unavailable are replaced by the system. Note that X can be (and as we'll see from the analysis that it actually is) typically much smaller than the total number of unavailable fragments at that instant.

The advantages of our randomized lazy strategy include:

(i) The repair process is continuous and adaptive, and does not have knee-jerk reactions. When fewer fragments are available, more repairs take place, while when more fragments are available, fewer probing and repair operations are required. Thus this strategy repairs all object all the while a little bit adaptive to the rate of churn, unlike the deterministic procrastination Strategy-A, which repairs objects less frequently, but needs to do a lot of repair work every time it is repairing an object. We'll see in the following that such an approach makes Strategy-A much more vulnerable to both churn and correlated failures.

(ii) Reduction in the number of probe messages even though cost of probing is not that critical a load on the system.

Obvious extensions of the sampling random subsets based maintenance strategy will include self-tuning the threshold as well as adapting the probing period, but we do not investigate such variants here. Also note that the eager repair strategy is a special case of either of the lazy strategies (corresponding to $T_a = N$ or $T_b = N$).

In principle, if less than M fragments of an object are left in the system, there is no more guarantee that the object will persist in the system. However there are two obvious out of band mechanisms apart the normal maintenance operations. (a) Owner of the object or any other user who has previously used the object and has a local copy may reintroduce it in the system. (b) In any practical implementation of the maintenance operations, the prober(s) for a specific object will keep a local cache of the object in order to optimize the repair process, which can still be used to reintroduce the lost object as well.

However, particularly since we are looking at a scenario with high churn, we ignore these out of band mechanisms, since the original source may be (even permanently) away from the system, as well as over time, different peers will act as the prober of an object since the previous ones might be off-line.

Hence, we will look into only the resilience that is guaranteed by the stand alone maintenance schemes themselves.

4 Markovian time-evolution analysis

Existing literature on P2P storage systems under churn fail to use a proper analytical tool to objectively determine the performance of the system given a churn and any specific maintenance strategy. Thus, a proper theoretical framework to compare two strategies too is non-existent, leaving no recourse to the P2P storage system researchers apart resorting to simulations. For instance, Bhagwan et. al [4] looks into system designing where they introduced the deterministic lazy maintenance mechanism and evaluate the system performance based on simulation as well as prototyping, however they [4] do not provide any new analytical insight into the system's behavior under continuous churn and repair processes. Similarly Weatherspoon et. al's recent work [22] benchmarks several existing storage systems through simulation experiments for some specific churn levels, but does not delve into analysis of the systems' dynamics. This leaves an important void in the objective understanding of such storage systems' behavior under churn, despite an abundance of empirical results from simulations and prototypes [4, 10, 22].

In order to evaluate our maintenance scheme, we identify and develop the right analytical tools, and we validate the analytical predictions with simulations.

Churn has been more exhaustively studied in the context of peer-to-peer overlay routing networks using various models - (i) Static resilience e.g., Gummadi et. al [9], (ii) Half-life e.g., Liben-Nowell et. al [13] and (iii) Markovian time-evolution analysis, e.g., Aberer et. al [1]. In the following we argue why the later is the only way to comprehensively compare maintenance schemes.

Static resilience: The basic question answered using a static resilience analysis is: "If a certain fraction of the peers have left the network and a corresponding fraction of information is unavailable, what is the performance of the system?"

All existing analyses of P2P storage systems at best fall into this category [3, 23] (also reused in [4, 10, 20]). Weatherspoon et. al [23] look into permanent disk failures as the dominant model for unavailability, and hence completely ignore temporal effects of churn. Thus it is actually Bhagwan et. al [3] who investigate the static resilience of the system. This model essentially looks at a snap-shot of the system, completely ignoring any new failures or repairs. Since this model does not at all look into the repair/maintenance process it is not suitable to compare different maintenance schemes.

Δ-life: This model looks into the lower bound of the cost that is needed to completely repair the system essentially answering he following: As the network membership changes over time, such that only Δ (say 0.5) fraction of the original peers remain in the system, what is the minimal number of repairs that are required to restore back to a fully consistent state (get all information up to date)? This model too does not look into the performance of any given maintenance scheme.

Time-evolution: Dynamics of large probabilistic systems is traditionally studied using Markov chain [11]. Any change (event) in the system is considered as a transition from one possible system state to another. We adapt such a well established methodology in the context of studying the dynamics of P2P storage systems under churn - by modeling the system as a Markov process and looking into the time evolution of the probability density function of all the possible states the system can be in, and hence to see if this distribution function converges to a *steady state/dynamic equilibrium* in the long run. If such a steady state exists, then it determines the operational state of the system under the given churn and adopted maintenance strategy, which in turn is necessary to determine the performance vs. operational cost trade-offs in the system.

We evaluate and compare our proposed randomized maintenance scheme with the existing deterministic maintenance scheme by studying the time-evolution of the system for both of these maintenance schemes. We also validate our analysis with simulations.

In that respect, apart proposing a better maintenance scheme, we also employ the time-evolution analysis in the context of P2P storage systems. Having such an analytical model has several other benefits. (a) Wider parameter ranges can be explored accurately much faster than running experiments. (b) Unlike simulation results which are vulnerable to implementation artifacts and whose interpretation is essentially open to speculation, the analysis provides a precise cause-and-effect picture of the system dynamics.

Its worth mentioning that even for the existing empirical studies [4, 10, 22], where the information must have been available, no one looked into the frequency distribution of the system states but only at the mean value. This is possibly a consequence of the fact that without a proper abstraction (as is required for the analysis), even obtainable information has been ignored in the existing literature, simply because it was not well understood as to what to look for and how to use this information.

5 Churn model

We model churn according to an exponential lifetime distribution for each online session of any peer as well as the period a peer stays off-line (for a given total peer population). Thus we assume that irrespective of the history at any time instant t, an online node will become unavailable with a probability $\delta_{\downarrow}$ at time $t+1$. Similarly an offline peer will rejoin the system with all its locally stored content at time $t+1$ with probability $\mu_{\uparrow}$. The fraction of available (online) peers is then given by $p_{on} = \frac{\mu_{\uparrow}}{\mu_{\uparrow}+\delta_{\downarrow}}$. Only this average availability is used in existing analysis [3] to study the effect of churn on static resilience in storage systems. Instead we take into account the dynamicity of the system, particularly studying its time evolution using a Markov model.

New peers joining the system will also bring additional storage space (and new objects to be stored), however these peers would not "bring back" the lost fragments, and hence not explicitly accounted for in the analysis. So to say, even if $\mu_{\uparrow} = 0$, the overall network size may stay stable or shrink or even expand, depending on the rate of new peer arrivals. While these new peers do not bring back missing fragments, the continuous maintenance operation will of-course exploit these new peers storage space while replenishing lost redundancy. Similarly peers departing permanently from the network will no more influence the system or its dynamics. Thus the above expression for the parameter p_{on} needs to be understood in the context: roughly speaking, it is the average availability of the existing peers in the network for a mid-term future (in comparison to the period for maintenance operations). Thus, the implicit assumption is that even if all the current peers eventually leave the network in the long run, the maintenance operation will in the mean-while replace the stored objects at the newly arriving peers. This in turn implicitly assumes that there is actually sufficient storage space in the network. If the network capacity does not increase over time (which is likely since there will be only limited number of nodes, each with storage limitations) but the content volume increases (with proliferation of various devices to produce huge volume of digital content this is also quite likely), the overall storage capacity of the network can become a bottleneck. This is however a more general problem for p2p storage systems and applications designer and is completely orthogonal to the focus of this paper where we only look at the impact of continuous and simultaneous churn and maintenance operations on the performance, and specifically availability/durability of the stored content. One practical way to deal with exhaustion of storage capacity is to lease storage space for a specific time-span and the storage layer provides availability and persistence of the stored object only for this lease-period, after which the object is gradually garbage collected.

Since the existing maintenance operations (whichever strategy) are invoked periodically, the real time is disentangled from the analysis - more frequent maintenance operations will mean lower perceived churn and vice-versa. Though not done by existing systems like Glacier [10] or TotalRecall [4], one can expect future generations of P2P storage systems to adapt the period of maintenance operations adapted to actual churn conditions, as well as differentiating the importance of various stored objects and various application requirements (while using the same underlying storage system). In these terms, such a disentanglement from the real time provides us the right abstraction, so that the analysis stays generic and only the system parameters $\mu_{\uparrow}$ and $\delta_{\downarrow}$ would be different for different scenarios. This approach is similar in spirit to physicists' use of intensive variables (i.e., scale invariant metrics) to study large scale systems, and has also been used in studying the properties of overlay routing networks under churn [8].

There may be some concerns with the churn model we use, but we argue why these are not critical: (i) This model does not look into the effect of permanent departure of nodes from the system nor new peer joins. In the context of storage systems, new peers joining the system do not change availability of already existing objects. But if we assume that the mean session time of a peer is relatively smaller than its life-time in the system, then the availability is threatened more by temporary departures. Since the system's repair mechanism will replenish the lost redundancy, we assume that relatively infrequent permanent departures do not influence the system's availability, particularly since the maintenance scheme will use other peers to compensate for the gradual permanent departures. Such an assumption is justified by measurement studies (such as [2]) on mean life-time and session-time. (ii) The rate of churn itself varies over time. In such a situation, the system continuously tries to converge to the corresponding steady state, and hence our simplistic analysis continues to provide a holistic insight into the system's behavior - particularly its stability and performance.

6 Analysis: Erasure code based redundancy, lazy maintenance

In order to quantify the performance and compare maintenance schemes, we are typically interested in: *"What is the operational cost of such a system vis-à-vis its resilience?"*

To answer this, we need to better understand the system's dynamics: *"What is the ensemble state of the system (e.g., the time evolution of the probability density function of actual redundancy of the stored objects)?"* and *"Whether it converges to a steady state?"*.

We'll restrict our study to only finite rate erasure code based redundancy (*M out-of N* erasure code). Analysis for other redundancy mechanisms - replication and rateless (Digital Fountain) erasure codes - as well as more sophisticated strategies, e.g., with self-tuning probing periods and repair thresholds, remain as part of our future work, but will rely on the same analysis methodology.

Implicit assumptions: The probing is done according to the maintenance strategy periodically, represented as discrete time t. As previously mentioned, the churn in the system is defined by two parameters: $\delta_{\downarrow}$ and $\mu_{\uparrow}$, representing the probability that an online peer goes off-line or an off-line peer rejoins the system between time t and $t+1$. Fluctuations because of any peer going offline and returning within this period (or vice-versa) is of-course transparent to the maintenance operation. Furthermore, we assume that the maintenance operation of different objects is synchronized, and the whole system goes cyclically through two distinct phases: churn and maintenance. Beside the simplification of the analysis, such an approach provides a modular model discerning the separation of concerns in the analysis, such that we obtain two sets of equations: one dependent solely on the churn model, another dependent only on the maintenance strategy, hence making them reusable in the overall analysis when only one aspect of the system (say maintenance strategy) changes. Time for reconstruction of fragment is ignored in our model. This can induce a physical limitation on the repair period which in turn will naturally lead to a minimum amount of churn the system will have to tolerate irrespective of however aggressive a repair strategy one may wish to implement.

Notation and terminology: We say that an object is in state i at any given time if i out of the N erasure encoded fragments of the object are available at that given time. We define $S_i(t)$ as the probability that i out of the N possible fragments of any object are online at time t just after the maintenance operations. $\widetilde{S}_i(t)$ is the probability that i fragments of any object are online at time t just before the maintenance operations (and after churn since time $t-1$). $\sum_i S_i(t) = 1$ and $\sum_i \widetilde{S}_i(t) = 1$ are the standard normalization for probability distributions. We also define $\widehat{S}_i(t) = (S_i(t) + \widetilde{S}_i(t))/2$ as the average of these two states. In reality, since churn is continuous and repair of different objects can not be synchronized, nor is it necessary and in fact undesirable since its better to use the network all the while to maximize its use, we expect that when repairs are indeed not synchronized, the system will actually reside in this state instead of oscillating between the two artificially introduced (before and after churn/maintenance) phases for analytical simplification and modularity. We revert back to this issue while validating our analysis in Section 7 (particularly Figures 1(c) and 1(d)).

Note that the whole system state is defined by these variables (S_i,$\widetilde{S}_i$). It does not matter how the system reaches a specific state till any time t, the system's time evolution from this point can then be modeled as a Markovian process. In particular, a recursive relationship between $S_i(t)$s and $\widetilde{S}_i(t)$s can be defined as follows.

6.1 Effect of churn

Irrespective of the maintenance mechanism in use, because of churn (our specific model parameterized by $\delta_{\downarrow}$ and $\mu_{\uparrow}$) we obtain the following recursive relationship.

$$\begin{aligned}
\widetilde{S}_i(t+1) &= S_i(t) \\
&\quad -S_i(t)\left[\sum_{l=0}^{i}\begin{pmatrix} i \\ l \end{pmatrix}\delta_{\downarrow}^{l}(1-\delta_{\downarrow})^{i-l}\left(\sum_{g=0;g\neq l}^{N-i}\begin{pmatrix} N-i \\ g \end{pmatrix}\mu_{\uparrow}^{g}(1-\mu_{\uparrow})^{N-i-g}\right)\right] && (1)\\
&\quad +\sum_{j=0}^{i}\sum_{l=0}^{j} S_j\begin{pmatrix} j \\ l \end{pmatrix}\begin{pmatrix} N-j \\ g \end{pmatrix}\delta_{\downarrow}^{l}(1-\delta_{\downarrow})^{j-l}\mu_{\uparrow}^{g}(1-\mu_{\uparrow})^{N-j-g}\ \textbf{where g=i-j+l} && (2)\\
&\quad +\sum_{j=i+1}^{N}\sum_{g=0}^{Min[N-j,i]} S_j\begin{pmatrix} j \\ l \end{pmatrix}\begin{pmatrix} N-j \\ g \end{pmatrix}\delta_{\downarrow}^{l}(1-\delta_{\downarrow})^{j-l}\mu_{\uparrow}^{g}(1-\mu_{\uparrow})^{N-j-g}\ \textbf{where l=j-i+g} && (3)
\end{aligned}$$

In the equation above, the term (1) represents the outflow from state i because of churn. This happens for any object in state i when any l of its i online fragments go off-line, and any $g \neq l$ of its $N-i$ off-line fragments come online. The term (2) is the inflow into state i from states $j \leq i$, where the number of fragments for the corresponding state j that go offline (l) and the number of fragments that come back online (g) are mutually related such that $g = i - j + l$. The corresponding object ends up into state i from states $j \leq i$. When $i < j$, similar combinatorial arguments hold - term (3). In addition, $i - g = j - l \geq 0 \Rightarrow g \leq i$ and $g \leq N - j$ determine the possible values of the number of fragments coming online (g) from states $j > i$ which can still cause inflow into state i because of simultaneous losses. The corresponding loss l is mutually related to g such that $l = j - i + g$.

6.2 Lazy Maintenance Strategy-A: Deterministic Procrastination

We have the following recurrence relationship for the deterministic procrastination based lazy maintenance Strategy-A introduced earlier in Section 3. We consider T_a to be the threshold defined in this maintenance strategy.

$$S_N(t+1) = \widetilde{S}_N(t+1) + \sum_{j=M}^{T_a}\widetilde{S}_j(t+1) \tag{4}$$

For $T_a \leq i < N$, $S_i(t+1) = \widetilde{S}_i(t+1)$, while for $M \leq i < T_a$ we have $S_i(t+1) = 0$ since if there are more than $M-1$ but less than T_a fragments available, all fragments will be repaired, and for $i < M$ we have $S_i(t+1) = \widetilde{S}_i(t+1)$ since normal repair operations can not reproduce and repair a data with fewer than M of its fragments available in the system. Some out-of-band mechanism to reintroduce the fragments, like by the owner of the object, is beyond the scope of this analysis. Some objects will always go to such states with a positive, even if very small probability. Thus, there is a "leak" in the probability mass, such that eventually all objects will be lost, unless we consider existence of such an out-of-band mechanism. Thus we normalize the probability distribution in each time round, compensating for the small nonetheless finite loss. This normalization process can also be viewed as if the probability distribution function we obtain from the analysis corresponds to the probability distribution corresponding to only the available objects in the system. Despite such a "*trick*", our analytical model successfully captures the system behavior as validated in subsequent simulations. Particularly refer to the discussions in Sections 7.3 and 7.5.

The cost of repair operations per object per repair period at a time t is then:

$$C_a^r(t) = \sum_{j=M}^{T_a}\widehat{S}_j(t)(N-j) \tag{5}$$

We use $\widehat{S}_i = (S_i + \widetilde{S}_i)/2$ for calculating the cost since in a realistic setting with non-synchronized repair operations for different objects, the system will reside in such a mean state. Though, alternatively one may well use $\widetilde{S}_i$ in order to obtain a more pessimistic estimate of the costs.

Cost of probing per object per repair period is $C_a^p(t) = N$.

6.3 Lazy Maintenance Strategy-B: Sampling Random Subsets

For this case we have the following recurrence relationship. We assume T_b as the threshold defined in this strategy.

$$S_N(t+1) = \widetilde{S}_N(t+1) + \sum_{r=1}^{N-T_b} \widetilde{S}_{N-r}(t+1)P_{N-r}(x=r) + \sum_{j=M}^{T_b-1} \widetilde{S}_j(t+1) \tag{6}$$

For $T_b \leq i < N$:

$$S_i(t+1) \quad = \quad \widetilde{S}_i(t+1) - \widetilde{S}_i(t+1)\sum_{r=1}^{N-i} P_i(x=r) + \sum_{r=1}^{i-T_b} \widetilde{S}_{i-r}(t+1)P_{i-r}(x=r) \tag{7}$$

where $P_i(X = j)$ is the probability that $T_b + j$ fragments are randomly (and sequentially) accessed in order to find T_b available fragments (after which the probing is stopped in that time round), when i out of the possible N fragments are actually available.

$$P_i(X=j) = \tbinom{j+T_b-1}{j}\frac{i!}{N!}\frac{(N-T_b)!}{(i-T_b)!}\frac{(N-i)!}{(N-i-j)!}\frac{(N-T_b-j)!}{(N-T_b)!} \tag{8}$$

This expression comes about because, of the first $T_b + j - 1$ fragments probed exactly $T_b - 1$ must be online (the $T_b{+}jth$ fragment probed is also online, which is why the probing terminates). They might have been probed in any interleaved sequence along with the j offline fragments probed. There are $\begin{pmatrix} i \\ T_b \end{pmatrix}$ possible ways of choosing the T_b live fragments out of a total of $\begin{pmatrix} N \\ T_b \end{pmatrix}$ ways of choosing T_b fragments. Similarly, the j off-line fragments are chosen from the $N - i$ off-line fragments, while they could actually have been chosen from any of the other $N - T_b$ fragments.

For $M \leq i < T_b$ we have $S_i(t+1) = 0$ since if there are more than $M - 1$ but less than T_b fragments available, all fragments will be repaired and for $i < M$ we have $S_i(t+1) = \widetilde{S}_i(t+1)$ since normal repair operations can not reproduce and repair a data with fewer than M of its fragments available in the system.

The cost of repair operations per object per repair period at a time t is then:

$$C_b^r(t) = \sum_{j=M}^{T_b} \widehat{S}_j(t)(N-j) + \sum_{j=T_b+1}^{N-1} \widehat{S}_j(t) \sum_{r=1}^{N-j} rP_j(x=r) \tag{9}$$

Cost of probing per object per repair period can be given as[2]:

$$C_b^p(t) \quad = \quad N\sum_{i=0}^{T_b}\widehat{S}_i(t) + \sum_{i=T_b+1}^{N}\sum_{j=0}^{N-i}(T_b+j)\widehat{S}_i(t)P_i(x=j) \tag{10}$$

[2]Note that the actual cost of probing is in any case negligible in comparison to the cost of fragment replacement is thus not critical. Also if more than T_b fragments are available, the cost of probing is $T_b + C_b^r(t)$.

6.4 Correlated failures

Apart from being resilient to regular and continuous churn, a persistent storage system should also be able to deal with rarer nonetheless inevitable correlated/catastrophic failures.

A way to model correlated failures is to assume that a fraction f_{corr} of the peer population are affected by the correlated failure. However, since the different fragments of the same object are stored at randomly chosen peers, each fragment is lost because of the correlated failure with a probability f_{corr} independently of each other. This model for correlated failure has been used in studying Glacier [10].

In such an event, an object which had i live fragments available before the correlated failure affecting f_{corr} fraction of peers will survive the correlated failure with a probability $\sum_{j=0}^{i-M} \left(\begin{array}{c} i \\ j \end{array} \right) f_{corr}^{j}(1 - f_{corr})^{i-j}$. Thus the overall probability for a single object to survive a correlated failure while using a specific lazy maintenance scheme under normal churn is given as:

$$\mathcal{D}_1 \;=\; \sum_{i=M}^{N} \widehat{S}_i(t) \sum_{j=0}^{i-M} \left(\begin{array}{c} i \\ j \end{array} \right) f_{corr}^{j}(1 - f_{corr})^{i-j} \qquad (11)$$

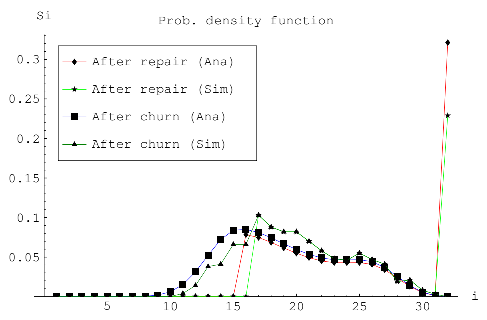
Finally, an end-user of such a storage system storing x objects will be concerned about not only the persistence of a single object with high probability, but also that none of the x objects are lost $\mathcal{D}_x = (\mathcal{D}_1)^x$ assuming that fragments for all the x objects are stored at different peers. Without this assumption, we'd have a higher probability of not losing any of the x objects, however, when objects will be lost, many objects will be lost simultaneously.

7 Results

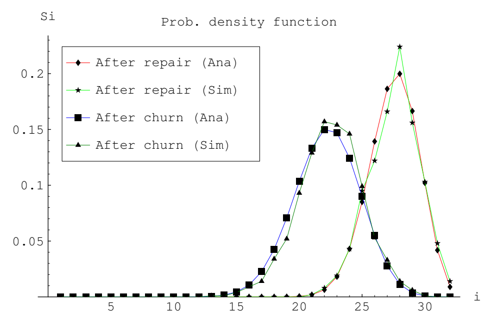
We validate our model with exhaustive simulations and briefly report some of the results here. We compare the two lazy maintenance schemes and observe that the sampling of random subsets based lazy maintenance strategy (Strategy-B) proposed by us has better performance than the deterministic procrastination based existing lazy maintenance strategy (Strategy-A). What is important in the simulations is to respect the independence of object fragments availability. This implies a large enough peer population, but that apart the peer population itself does not play any role. For the statistical properties to hold good, we need at least a moderate number of objects to do the averaging across these objects in order to determine an observed distribution function of the states of the objects. Unless otherwise specified, our default experimental setting for the results presented subsequently was as follows:

We considered 200 distinct stored objects. The simulations were run for 200 time units (maintenance cycles) though the steady state is approached within a much shorter time span. Moreover the observed distribution is bound to fluctuate a bit from one time round to another, thus we average the simulation results over a time window of 5 time units. Even for such a small time-window for averaging led to a fairly stable probability distribution over time, demonstrating the low deviation of the system from the steady-state. We use a *8 out-of 32* (rate 0.25) erasure code. We used $T_a = 16$ in *Strategy-A* and $T_b = 12$ in *Strategy-B*. The experiments were conducted for both settings: synchronized as well as the more realistic non-synchronized repair cycles (but the same periodicity) for different objects. For churn, we typically used $\delta_{\downarrow} = 0.2$ and $\mu_{\uparrow} = 0.1$. The results obtained from the simulations matched well with the prediction from our analysis.

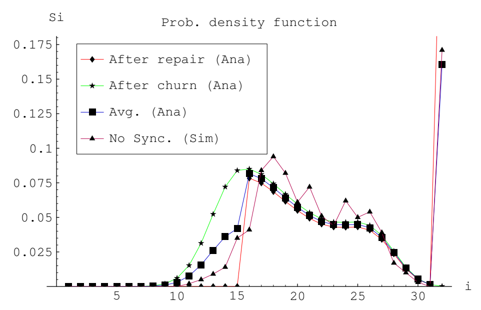
7.1 Validation of the analytical model



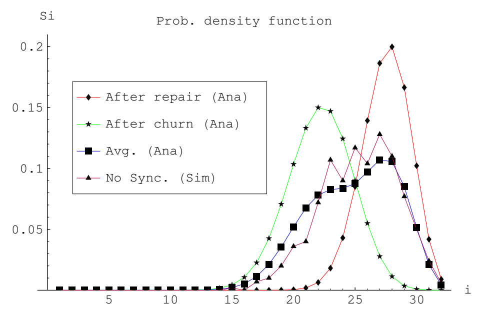
(a) Simulation with synchronization of repairs (Strategy-A)



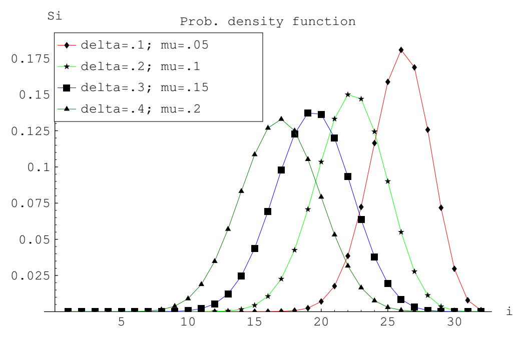
(b) Simulation with synchronization of repairs (Strategy-B)



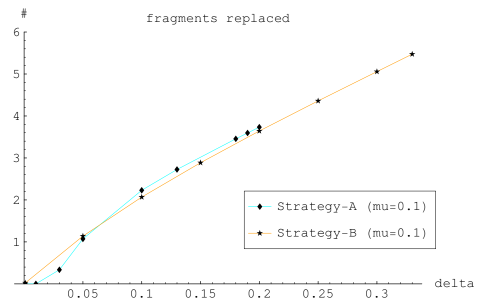
(c) Simulation with no synchronization of repairs (Strategy-A).



(d) Simulation with no synchronization of repairs (Strategy-B).



(e) $\widetilde{S}_i$ for different $\delta_{\downarrow}$ and $\mu_{\uparrow}$ for same $p_{on} = 1/3$ (Strategy-B)



(f) Maintenance overhead (fragments replaced) per maintenance cycle

Figure 1: Simulation based validation of the analytical model (a,b,c,d) and some analytical results (e,f)

Based both on our analysis (equations solved numerically) and simulations, we observed that the probability distribution functions $S_i(t)$ and $\widetilde{S}_i(t)$ converge over time (and in fact fast) to the steady-state values, demonstrating that all other things being same, particularly the parameters determining the churn, the system indeed converges to a steady operational state[3]. We show some results from our analysis and experiments for deterministic procrastination *Strategy-A* (*Figures 1(a),1(c) and 1(f)*) and sampling

[3]Thus we'll use only the steady-state distributions S_i and $\widetilde{S}_i$ in the following, without anymore referring to the time t.

random subsets *Strategy-B* (*Figures 1(b), 1(d)*, 1(e) and 1(f)). The *x-axis* in the plots corresponds to the states i - the number of available fragments for any object out of the possible N. In Figures 1(a) to 1(e) the *y-axis* shows the probability mass associated with the corresponding states, just after repair operations are performed S_i, and just before maintenance operations are performed $\widetilde{S}_i$ i.e., after the churn phase. Figure 1(e) shows only $\widetilde{S}_i$ for various churn levels. A first setting of our simulation adhered to the analysis model where the repair operations for all objects were synchronized. This led to the two distributions for each state - one after the repair phase and one just before it (Figures 1(a) and 1(b)), and we see that the analytical prediction of the system state concurs with the experimental results. In practice, such synchronization will not be realistic, and thus the repair operations (replacement of unavailable object fragments) as well as churn (loss or regain of object fragments) will be continuous and randomly interleaved. We simulated the system where there's no synchronization of the repair operations for different objects, and compared it with the result obtained by averaging the two distributions obtained from the analysis. As can be seen from Figures 1(c) and 1(d), the simulation based result from the model without synchronization of repair operations matched very well with the average obtained from the analytical prediction (averaged).

These results validate that despite the simplifications, particularly with respect to the separation of concern of the effects of churn and repairs, we have an appropriate analytical model capturing the system dynamics.

7.2 Static resilience versus steady state analysis

Previously we had noted that $p_{on} = \frac{\mu_\uparrow}{\mu_\uparrow + \delta_\downarrow}$ is the fraction of online peers, and hence corresponds to the average peer availability in the system. In Figure 1(e) we show the $\widetilde{S}_i$ analytically obtained for various $\mu_\uparrow$ and $\delta_\downarrow$ but the same $p_{on} = 1/3$ (using maintenance Strategy-B). The more the probability mass shifts leftwards (lower values of i), the more vulnerable the system is. From the figure its clear that even if the peers' average availability is the same, with higher churn (characterized by higher values of $\mu_\uparrow$ and $\delta_\downarrow$), the system is less robust. Such inferences on the system's dynamic resilience is not captured by the static resilience study [3], since it fails to distinguish two systems with same average behavior but different dynamics. The system's actual state in turn has pronounced implications, and the dynamic equilibrium analysis provides us a better glimpse of the system's inner working and hence its actual resilience.

7.3 Overheads of lazy maintenance mechanisms

Sampling of random subsets always needs less (or at most same) probes per object as the deterministic procrastination, which always probes for all fragments. However the probing cost is not a dominant cost in the system and hence we do not show it here. The replacement of fragments is however expensive.

In Figure 1(f) we show the average number of fragments that are replaced for a churn represented by the parameters $\mu_\uparrow = 0.1$ and varying $\delta_\downarrow$ for the two maintenance schemes. The actual effort in terms of consumed bandwidth and CPU usage will of-course depend on the size of the stored objects as well as the particulars of the implementation. In the plot we show for each maintenance strategies only the range of $\delta_\downarrow$ for which $\sum_{i=0}^{M-1} \widetilde{S}_i \leq 10^{-4}$. This condition guarantees that the availability of individual objects under the given churn and chosen maintenance strategy stays higher than 0.9999. The choice of the number 10^{-4} is arbitrary, and something else could have been chosen as well. However this number needs to be sufficiently small, both in order to ensure good availability guarantee of stored objects, as well as to ensure that the normalization argument used in Section 6 is rational. We'll revert back to this issue also in Section 7.5 while explaining results corresponding to Figure 3.

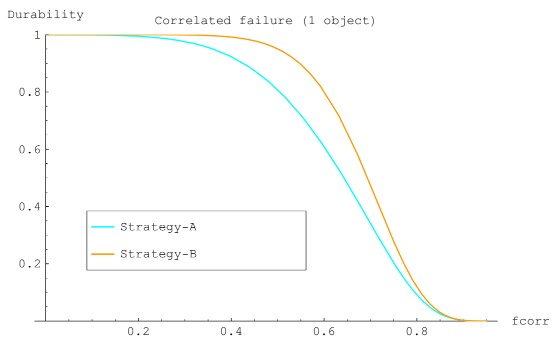
From Figure 1(f) we can infer two things. First of all, while the two maintenance strategies have very similar average cost of repairing fragments per maintenance round, deterministic procrastination has somewhat lower overheads only at very low churn rate ($\delta_\downarrow$) while mostly sampling of random subsets has lower average fragment replacement overhead. In fact the maintenance scheme parameters T_a and T_b for these results were chosen such that the two maintenance schemes have comparable expected repair overheads, so that the resilience achieved can then be compared.

This is because once the churn is high enough, repairs are triggered frequently enough by the deterministic procrastination based scheme, it is just that it does the repairs for each object in impulses - repairing between $N - M$ to $N - T_a$ fragments for the same object in a single maintenance round, because it first lets many fragments to become unavailable. In contrast, the sampling of random subsets based mechanism naturally has to sample and repair more fragments for high churn and fewer for low churn (sample size is determined by the probability distribution as determined in Equation 8). Since the repair process for each object is continuous, that is, some of the unavailable fragments are replaced in each maintenance round, the same effort is more evenly distributed over time per object.

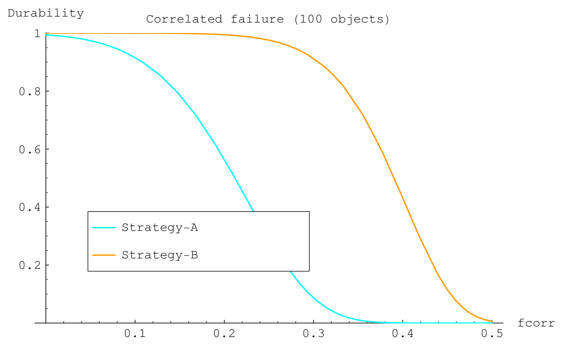
A consequence of such a continuous but lower effort per object per round is that almost all objects always retain much better redundancy, that is, the system is "*healthier*" and has a better resilience against churn. Thus for same $\mu_\uparrow = 0.1$, Strategy-A guarantees a 0.9999 availability for each object only for $\delta_\downarrow \leq 0.2$ while Strategy-B tolerates churn till $\delta_\downarrow \leq 0.33$.

Apart from higher resilience against normal churn, this also has implications on the robustness of the system against correlated faults, as is discussed next.

7.4 Surviving correlated failures while using lazy repairs



(a) Durability of any single object ($\mathcal{D}_1$)



(b) Durability of all "100" objects ($\mathcal{D}_{100}$)

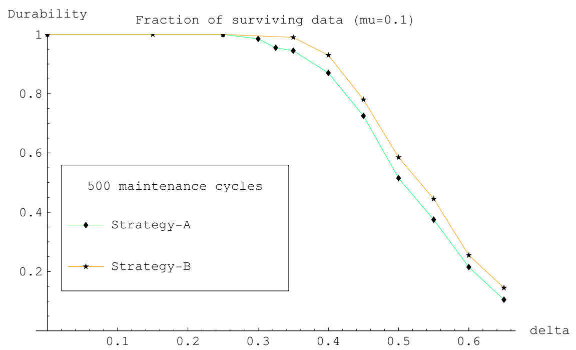
Figure 2: Durability under correlated failure f_{corr} in addition to regular churn ($\mu_\uparrow = 0.1$, $\delta_\downarrow = 0.2$)

Glacier [10] uses a proactive repair strategy and high redundancy to deal with normal churn. Proactive strategies, particularly for large objects (which is exactly where use of erasure codes make sense) however have prohibitive maintenance cost, which motivated the use of a lazy maintenance scheme in

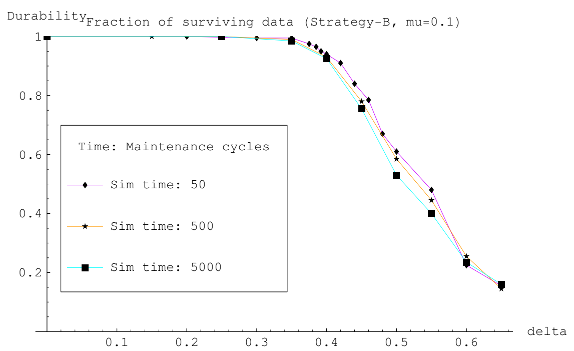
TotalRecall [4]. However, the deterministic procrastination (Strategy-A) used in TotalRecall leaves it very vulnerable to even a small degree of correlated failures, an aspect not accounted for in its design (nor evaluated). In contrast, the randomized sampling based maintenance Strategy-B we introduced in this paper, while having the benefits of being lazy also provides much better resilience against correlated failures. A lazy mechanism can never compete with an eager one in terms of resilience, but still, the randomized lazy maintenance mechanism provides a much a better compromise between maintenance cost and resilience, unlike the deterministic procrastination based scheme which has marginal tolerance against correlated failures. In Figure 2 we show the durability of any individual object, as well as the durability of a collection of 100 objects (that is, the probability that none of these 100 objects are lost) for the two lazy maintenance schemes.

The stark difference in resilience of the two lazy maintenance schemes despite similar repair costs (under regular churn) is readily explained from the the probability distributions $\widehat{S}_i$ corresponding to the two maintenance strategies as observed in Figures 1(c) and 1(d). Deterministic procrastination allows a large number of objects to concentrate close to the state T_a, while randomized sampling keeps the system far away from the edge even while spending comparable maintenance effort even for a smaller T_b (than T_a) during normal churn. This essentially means that using the randomized sampling based scheme, the system has a much better "*health*" and hence has greater resilience against regular churn as well as occasional correlated failures.

7.5 Convergence, uniqueness and stability (of the system) and validity of the model



(a) Comparison of the two maintenance strategies



(b) Simulation run for 50, 500 and 5000 maintenance cycles

Figure 3: Threshold (phase-transition) behavior observed by simulations.

Finally, we show some simulation results to look into the fraction of data that survive normal churn as well as investigate the settings for which our model is appropriate, and when the simplifying assumptions that we made no more hold. The simulations are for $\mu_{\uparrow} = 0.1$ and varying $\delta_{\downarrow}$. The x-axis in these plots is $\delta_{\downarrow}$, and the y-axis is the durability of the objects - fraction of objects which stay available for the whole simulation period. For each setting, the experiments were conducted 5 times, and the worst performance was chosen. Moreover, even as durability and availability are strictly different, we considered the worst availability during the whole simulation as the indicator for the durability under the premise that if a specific object is once unavailable, it may never be recovered using the maintenance schemes themselves (even though some re-joining nodes may make the object available again).

We observe in Figure 3(a) that Strategy-B is more robust (tolerates larger $\delta_{\downarrow}$) than Strategy-A. We also run the experiments for various period, measured in terms of maintenance cycles, to ensure that the system's behavior is correctly captured. In Figure 3(b) we show results corresponding to the use of Strategy-B for various simulation durations.

From these results shown in Figure 3 we also observe that the system exhibits a threshold (*phase-transition*) behavior at a certain churn value. Beyond this threshold which depends on the maintenance scheme, the system is unstable, and given the chosen maintenance scheme and the churn conditions, stored objects would be lost. The analyses presented earlier in this paper cease to hold good beyond this threshold simply because the transition probabilities and the *normalization trick* (described in Section 6.2) does not hold good anymore in practice. Thus to say, we speculate that the same threshold determines the breaking point of the actual system as well as of the analytical model. However, from the perspective of most applications, the region of interest is indeed before the threshold (so that no object is lost) and the analysis provides the exact behavior of the system in this desirable zone of operation. Moreover its easy to estimate the correctness of the analysis based on the obtained result as already also explained in Section 7.3. In fact the condition "$\sum_{i=0}^{M-1} \widetilde{S}_i \leq 0.0001$" used there to make sure that the availability of an individual object is more than 0.9999 is a rather conservative estimate (so far as the validity of the analytical model is concerned), as the simulations show that the threshold is close to but somewhat larger than the churn levels considered there. The fact that the threshold observed in the simulations are indeed close also validate our approach of using $\sum_{i=0}^{M-1} \widetilde{S}_i$ as a metric to judge the breaking point of the system, hence making the analytical model itself useful (without having to simulate always to validate results and verify whether the system is stable or not). Both out of academic interest as well as a more precise estimate, we look forward to extend the present theory in order to hopefully analytically derive the threshold point. Determining analytically the precise threshold may require ideas and tools from percolation theory, but admittedly, that is just a speculation at this juncture.

Simulations also showed that starting from various arbitrary initial conditions (where data objects were still available), and given a churn and maintenance scheme, the system always converged to the corresponding unique dynamic equilibrium state. This also means if churn changes over time, the system will try to move from the dynamic equilibrium state to a new one corresponding to the new churn rate.

8 Ongoing and future work

As pointed out in the previous section, it'll be interesting to be able to predict precisely and analytically the breaking point of the system. We speculate the need of percolation theory in order to derive the critical churn corresponding to a maintenance strategy. However, as discussed in Section 7.5 the current theory already provides (heuristically using $\sum_{i=0}^{M-1} \widetilde{S}_i$) an approximate estimate, which is already a good indicator for systems design.

We hope that as a direct implication of this work, systems like TotalRecall can benefit by using the randomized sampling based maintenance scheme.

In the meanwhile, we are developing a separate storage system (tentatively called Digit4), where we use our randomized subset sampling based lazy maintenance scheme for large objects. That apart, Digit4 uses Digital Fountain (rateless) erasure codes. This has several practical benefits. First of-all, we do not need to keep track of which specific fragments are lost and replace precisely the same one, but only need to replace same number of fragments. Digital Fountain codes ensure that these new blocks introduced will be unique with respect to the previously inserted fragments. This also means that even if we do unnecessary repairs and fragments do come back, we'll only have increased redundancy. If finite rate erasure codes are used there will be duplicates of the same fragment (thus not adding any extra diversity), which in fact makes management and garbage collection tasks more complicated. With the use of Digital Fountain codes as our chosen erasure code and using the randomized sampling based maintenance scheme, we thus aim to realize a storage system with good resilience and maintenance cost as well as lower implementation complexity.

9 Conclusion

We proposed a randomized lazy repair strategy, which has much better performance in terms of resilience against churn and correlated failures for comparable (and mostly lower) repair costs in comparison to the existing lazy (deterministic procrastination) strategy.

It is relatively simple to determine the static resilience of a system [3], and was an important first step while choosing design parameters for a system. Static resilience however does not properly capture the dynamics of the system, nor give a clear picture of its resilience and performance under continuous churn and repair operations. In fact, as we observed, for the same average system state, the particulars of the dynamics can still greatly influence the system properties. Since static resilience completely ignores the dynamics, it is not useful for comparing different maintenance schemes.

Only studying the time evolution of the system, particularly measuring the probability mass/distribution function of the possible states gives a precise quantification of the system properties. We use Markovian time-evolution analysis based on which we observed that the system arrives at a steady state. We employed this analysis methodology to do a case study by comparing the two lazy maintenance schemes and evaluate precisely the performance and costs. We also validated these results with simulation experiments.

From system's perspective, the immediate implication is that the randomized lazy maintenance scheme we proposed here can be easily integrated to existing systems, given its simplicity as well as performance benefits.

References

[1] K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004.

[2] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[3] R. Bhagwan, S. Savage, and G. M. Voelker. Replication Strategies for Highly Available Peer-to-Peer Storage Systems. Technical Report CS2002-0726, UCSD, 2002.

[4] R. Bhagwan, K. Tati, Y-C. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *Proceedings of Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[5] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, 2003.

[6] L. Cox and B. Noble. Pastiche: Making backup cheap and easy. In *In Proceedings of Fifth USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, 2001.

[8] Sameh El-Ansary, Erik Aurell, and Seif Haridi. A physics-inspired performace evaluation of a structured peer-to-peer overlay network. In *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2005)*, 2005.

[9] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM Press.

[10] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[11] F. Heylighen and C. Joslyn. Cybernetics and second order cybernetics. *R.A. Meyers (ed.), Encyclopedia of Physical Science & Technology (3rd ed.), (Academic Press, New York)*, 4, 2001.

[12] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[13] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *PODC*, 2002.

[14] M. Luby. LT Codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[15] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, Mary Baker, and Yanto Muliadi. Lockss: A peer-to-peer digital preservation system. *ACM Transactions on Computer Systems (TOCS).*

[16] D.A. Patterson, G. Gibson, and R.H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of ACM Sigmod*, 1988.

[17] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[18] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. SIAM*, 8(2):300–304, June 1960.

[19] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The Oceanstore prototype. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[20] R. Rodrigues and B. Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *IPTPS*, 2005.

[21] A. Shokrollahi. Raptor codes, 2002.

[22] H. Weatehrspoon, B-G. Chun, C.W. So, and J. D. Kubiatowicz. Long-term data maintenance: A quantitative approach. Technical Report UCB/CSD-05-1404, UC Berkeley, 2005.

[23] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2001.

[24] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 90–106. IEEE Computer Society Press and Wiley, New York, NY, 2001.

Acknowledgment

We would like to thank Thomas Schwarz for the stimulating initial discussions which led us to look into the steady state behavior of erasure code based storage systems. We'd also like to thank Hakim Weatherspoon for briefing us with related developments in Oceanstore [12] and related projects at Berkeley. We'd also like to thank Brighten Godfrey for the general discussions on reliability in distributed systems. Finally we'll like to specially thank Amin Shokrollahi for interesting discussions on erasure codes in general and properties of Digital Fountain codes [14, 21] in particular, as well as facilitating the use of Digital Fountain codes for the ongoing implementation. The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.