Proactive Replication in Distributed Storage Systems Using Machine Availability Estimation

Alessandro Duminuco
Institut Eurecom
Sophia Antipolis, France
duminuco@eurecom.fr

Ernst Biersack
Institut Eurecom
Sophia Antipolis, France
erbi@eurecom.fr

Taoufik En-Najjary
Institut Eurecom
Sophia Antipolis, France
ennajjar@eurecom.fr

ABSTRACT

Distributed storage systems provide data availability by means of redundancy. To assure a fixed level of availability in case of node failures, new redundant fragments need to be introduced.

Since node failures can be either transient or permanent, deciding when to generate new fragments is non-trivial. An additional difficulty is due to the fact that the failure behavior in terms of the rate of permanent and transient failures may vary over time. To be able to adapt to changes in the failure behavior, many systems adopt a reactive approach, in which new fragments are created as soon as a failure is detected. However, reactive approaches tend to produce spikes in bandwidth consumption.

Proactive approaches create new fragments at a fixed rate that depends on the knowledge of the failure behavior or is given as a parameter by the system manager. However, existing proactive systems are not able to adapt to a changing failure behavior, which is common in real world.

We propose a new technique based on an ongoing estimation of the failure behavior that is modeled by a network of queues. This scheme combines the adaptiveness of reactive systems with the smooth bandwidth usage of proactive systems. It can be considered as a generalization of the two previous approaches, in which the duality reactive or proactive becomes a specific case of a wider approach tunable with respect to the dynamics of the failure behavior.

1. INTRODUCTION

Peer-to-peer systems have received a lot of attention in recent years. In particular file sharing systems have been very popular. The key property of P2P systems is self-scaling, i.e. as more peers become part of the system not only the service demand increases but also the *service capacity*. P2P systems can be used to build various distributed applications. One such application that has attracted the attention of the research community is peer-based distributed storage [1, 2, 5, 13, 10, 7]. Such a system is only useful if it can provide **service guarantees** concerning the durability and the availability of the stored data.

Durability means that, once stored, data are never lost. **Availability** assures that data can be retrieved in any moment, i.e. they are always available.

Note that availability implies durability, while the opposite is not always true. A durable object, can indeed be unavailable at certain periods, when the node on which it was stored goes temporarily offline. Peers that are not available for a limited time, due to disconnections or transient failures, affect availability, while peers that permanently leave the system affect durability.

The ability to provide such service guarantees is strongly related to the variations in peer availability. When building a distributed storage out of "unreliable" components, *redundancy* is used to achieve the required service guarantees. Redundancy helps to keep the data available, in case of transient failures. To cope with permanent failures and to assure durability, one needs to monitor the amount of redundant data and insert new redundant data when needed [18].

Redundancy in storage systems can be achieved in two different ways: (i) Replication of the original data or (ii) parity encoding of the original data. For a given amount of redundancy, parity encoding offers a higher degree of reliability than replications. In this paper we assume that parity encoding is used which works as follows: The original object to be stored is divided in k chunks that are coded in $k + h$ **fragments** such that any k fragments are sufficient to reconstruct the original object. Moreover we assume that the creation of a new fragment, called **repair**, can be always performed as long as a minimum number of fragments is available. Note that such a scheme is rate-less, i.e. the number of coded fragments is not bounded in the design phase. An example of such a technique is Network Coding [8]. In

*CoNEXT'*07, December 10-13, 2007, New York, NY, U.S.A.

our analysis we fix at k the minimum number of available fragments needed for a repair, which corresponds to the object availability. This choice is quite conservative, but keeps our system more general.

In this paper we present and analyze a scheme that periodically infers the failure behavior of the peers in order to determine when repairs need to be triggered. Our aim is to maximize the smoothness of the repair rate and thus the smoothness of the bandwidth needed for these repairs. While smoothness has not received much attention in the literature, we believe that it is a very important metric, especially in P2P systems. Peers typically only have a limited network bandwidth, which is determined by their access link capacity. Additionally, they may deliberately limit the bandwidth budget allocated to a P2P application in order to avoid degraded performance of other activities. If the repair process operates in bursts, the spikes in network bandwidth may be not sustainable by the peers and the success of the repair process may be delayed or compromised. Therefore, one should try to smooth as much as possible the repair rate anticipating some of the repair work in order to prevent bursts to occur.

Generally, repair schemes assume the existence of an entity, distributed or centralized, that is capable of monitoring the number of available fragments and decide when to schedule the repairs. We rely on this assumption as well. The details of how such a monitoring device works is beyond the scope of this paper. Peers could, for instance, be organized in a hierarchy and run a gossip algorithm to compute the number of available fragments [11, 12].

Existing approaches for the repair of lost fragments are either **reactive** or **proactive**. Reactive schemes are able to follow changes in failure behavior of peers and provide availability, however at the expense of a global waste of resources or bursty use of them.

Proactive schemes use a constant repair rate and are able to smooth the resource consumption for repairs. However, to provide durability, proactive schemes need an *a priori* knowledge of failure behavior. In case of imprecise or wrong knowledge, durability may be compromised.

We argue that proactive and reactive schemes represent two specific cases of a more general approach that tunes its reactivity with respect to the expected stability of the peers. In limit cases, it may result in a purely reactive scheme, if the peer behavior does not follow any predictable pattern, or a fixed repair rate, if peer availability remains constant.

We want to draw the attention to one important issue: Any scheme that does a repair to replace a fragment stored on a node that became temporarily unavailable may do *wasted work*. Indeed if the newly created fragment was stored on a peer that later permanently leaves, it may disappear before the system needs it. In this case that fragment would be completely useless for both durability and availability.

The main contribution of this paper is a framework based on an ongoing estimation of the peer failure behavior. The rate R, at which repairs are performed, is periodically updated accordingly to the changes in statistical properties of failures. This framework is able to provide at the same time durability, adaptiveness and a smooth use of the resources.

The design of this framework requires the solution of an adaptive control problem, presented in section 3, which is based on a periodical estimations of the peer behavior. In section 4 we discuss the impact of the estimation time, while in section 5 we propose a hybrid reactive-proactive scheme, which improves the availability guarantees of the system. The proposed system is first validated in section 6 and then evaluated using experiments as described in section 7.

2. BACKGROUND AND RELATED WORKS

Providing availability or durability is a key issue to be addressed by any work on distributed storage. The classical solution employed in most of the early DHT-based systems is what is usually defined as a purely **reactive scheme**, also known in literature as *eager repair scheme*. These systems [7, 5, 15] make no distinction between transient and permanent failures and the fragments stored on peers that come back after a transient failure are not reintegrated in the system. This approach is extremely simple and effective, but it does not address at all the efficiency of the maintenance, which may overwhelm in certain conditions the capacity of the system and compromise data durability.

From traces of peer availability we know that temporary disconnections are much more frequent than permanent ones. Reintegrating fragments should significantly reduce the number of repairs needed.

There exist a number of reactive schemes that use reintegration. These systems are more complex since they need to track the disconnected peers and react selectively to disconnection events. Usually they accomplish this last task by means of a threshold. *Carbonite* [3] uses a repair threshold TH_L which corresponds to the minimal level of redundancy needed to face transient failures and provide availability. TH_L is considered as a lower-bound: any time the number of available fragments runs below TH_L a single repair is performed. This threshold-based reactive scheme is the cheapest one in terms of resources consumed, since only the repairs strictly needed are performed. *Total Recall* [2] uses a lower-bound threshold TH_L as well, but fixes also another threshold TH_H, which is the amount of redundancy that is initially inserted and that is restored when the system runs below TH_L. This means that

the system triggers multiple repairs at once to bring the number of replicas back to the initial number TH_H. This approach represents a first step towards a proactive approach, where part of the work is done in advance with respect to the real needs.

The work in [6] is one of the first to address the time evolution and the steady state characteristics of a storage system. It still uses a threshold TH_L but adopts a proactive approach called random lazy repair strategy, in which the number of repairs done increases as the system gets closer to the threshold TH_L. The main intuition behind random lazy repair is that if one waits until the threshold is reached before repairs are made, the occurrence of correlated failures may put in danger the durability of the objects. Besides, this may result in a very bursty use of the resources needed.

To our knowledge, *Tempo* [4] is the only one that is concerned about smoothing the bandwidth used for repair. *Tempo* argues that reactive systems tend to perform repairs in bursts, alternating periods of intensive bandwidth consumption with periods of inactivity. The spikes produced by such a behavior represent both an inefficient use of the resources and a danger for object durability. The idea in *Tempo* is to have a constant repair rate that is not correlated to the *instantaneous condition* of the system. This rate is fixed by the system administrator in terms of a bandwidth budget per node, and if properly chosen, is able to assure durability. The main weakness of this method is that there is no way to choose *a priori* the right repair rate. Even if the bandwidth is used as smoothly as possible, there is neither guarantee on the object durability nor on the optimality of the resource consumption, which may be a lot higher than what is strictly needed.

The present work, starting from the considerations made by *Tempo*, tries to develop a system that strives to meet three different objectives: provide data availability, consume an amount of resources comparable with the one consumed by reactive schemes, and *maximize the smoothness of the repair bandwidth needed*.

The closest work in this sense is [6], which, however, does not analyze the impact of its randomized algorithm on the smoothness of the bandwidth.

3. AN ADAPTIVE CONTROL PROBLEM

A generic non-adaptive scheme uses a repair rate R that is constant in time to match a target number of available fragments.

In this work, we aim to build an *adaptive control scheme* that is able to adapt the repair rate to the changes of the system behavior.

We depict our adaptive control scheme in figure 1. In this scheme, the repair rate is a time-dependent signal $R(t)$ determined by the *controller*, which in this case receives also an estimation of the system behavior performed by the *estimator*.

The *system* is defined by a set of parameters. Starting from the observation of $n(t)$, the *estimator* is able to estimate these parameters, which in turn are used by the *controller* to choose $R(t)$. Note that the estimator receives as input an additional information called *transition*, which signals the occurrence of a repair or of a reconnection.

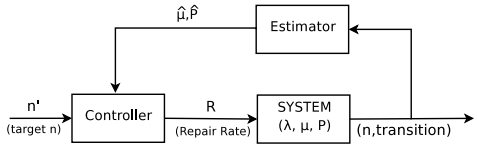


Figure 1: The adaptive control scheme.

The operations of estimation and control are performed periodically. We denote this period with ΔT, which represents both the estimation time, i.e. the observation period used by the estimator, and the update period, i.e. the interval between two updates of $R(t)$.

3.1 The System Model

The design of the estimator needs the definition of a mathematical model of the system, able to capture the behavior of peers.

First of all let us consider the life-cycle of a single peer belonging to the system. It can be represented by a state machine with three possible states. When the peer joins for the first time the system, it enters the *connected* state. After a session time, the peer may disconnect and enter the *abandon* state if it will never come back, otherwise the *temporarily disconnected* state. After a disconnection time, it will leave the *temporarily disconnected* state and reenter the *connected* state. This state machine corresponds to the Continuous-Time Semi-Markov chain depicted in figure 2, where the transition rates specified on the arcs are related to the following parameters:

- μ: **Single peer disconnection rate**. $1/\mu$ represents the average session duration.
- λ: **Single peer reconnection rate**. $1/\lambda$ represents the average time a peer stays temporarily disconnected before coming back online.
- P: **Abandon Probability**. When a peer disconnects it can go either temporarily or permanently offline. The actual event is governed by the probability P, which formally is the conditioned probability that a peer abandons the system given that it is leaving the *connected* state.

The behavior that describes the life-cycle of a single peer must be translated in a more complex model that

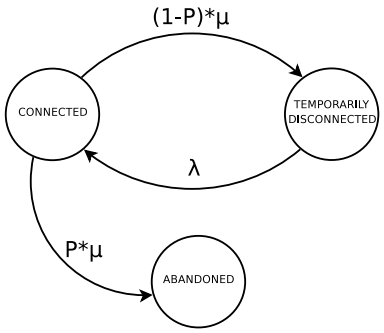


Figure 2: The Continuous-Time Semi-Markov chain of a peer life-cycle.

takes into account the participating peers all together and the repair rate R.

Given the above assumptions, we can use a network of two $G/G/\infty$ queues depicted in figure 3 to represent the system behavior. A $G/G/\infty$ queue represents a pure delay element where the delay, which corresponds to the service time, fits a generic distribution. In our case, Q_1 represents the peers in the *connected* state and Q_2 represents the peers in the *disconnected* state.

We assume that the number of peers is sufficiently large so that every fragment can be stored on a different peer. With this assumption the terms peer and fragment are equivalent and thus this model represents also the availability of the fragments in the system.

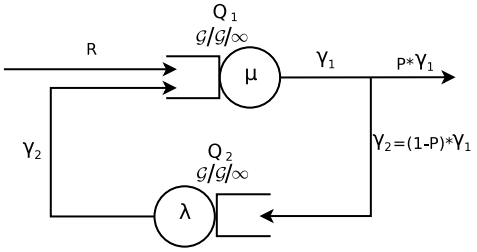


Figure 3: Queuing system representing the overall system behavior.

The customers of the first queue Q_1 represent the number of available fragments $n(t)$. Its arrival process, whose rate is denoted with γ_1, is given by the fragments that are newly introduced at rate R and the process of fragments that are becoming again online after a period of unavailability. The time spent in Q_1 is determined by the service rate μ. The departure flow from Q_1 represents the fragments becoming unavailable and its rate is equal to the arrival rate. The customers in the second queue Q_2 represent the number of temporarily unavailable fragments $m(t)$. Its arrival process, whose rate is denoted with γ_2, is given by the fragments that have become unavailable and with probability $(1-P)$ did not abandon permanently the system. The time spent in Q_2 is determined by the service rate λ. Finally, the departure process from Q_2 represents the fragments becoming available again.

We want to make a clarification concerning the two parameters μ and λ, which are defined, in the Continuous-Time Semi-Markov chain of figure 2, as the single peer disconnection and reconnection rates. In the rest of the paper, we will refer to them simply as **disconnection rate** and **reconnection rate**. In the queuing system of figure 3, μ and λ represent the service rates of the two queues and must not be confused with the global departure rates, which are referred to by γ_1 and γ_2.

We can solve this network of queues, writing its balance equations:

$$\begin{array}{rcl} \gamma_1 & = & R+\gamma_2 \\ \gamma_2 & = & (1-P)\gamma_1 \end{array} \Rightarrow \left\{ \begin{array}{rcl} \gamma_1 & = & \frac{1}{P}R \\ \gamma_2 & = & \frac{1-P}{P}R \end{array} \right. \quad (1)$$

Using Little's law [19] we can derive the average number of customers in each queue:

$$\overline{n} = \gamma_1/\mu = \frac{R}{\mu P} \qquad \overline{m} = \gamma_2/\lambda = \frac{(1-P)R}{\lambda P} \quad (2)$$

In case the service times and the arrival times are exponentially distributed, Q_1 and Q_2 become $M/M/\infty$ queues and we can write the analytical expression [19] of the probability distribution of the number n of available fragments as:

$$f(n) = \frac{\overline{n}^n}{n!} e^{-\overline{n}} \sim \mathcal{N}(\overline{n}, \overline{n}) \quad (3)$$

Note that eq. (3) can be approximated by the Normal distribution with a variance $\sigma^2 = \overline{n}$. This particular model will be used exclusively in the experiments with synthetic data to validate the system and to get some insights into its functioning.

3.2 The Estimator

The estimator is an object that, collecting statistical information on the signal $n(t)$ and on the input flow in the first queue, is able to estimate the parameters μ and P needed by the controller. We obtain:

- The average number of available fragments $\hat{n}$:

$$\hat{n} = \frac{\sum_i n_i t_i}{\Delta T} \quad (4)$$

  where t_i is the time spent by the system in the state n_i, which implies $\Delta T = \sum_i t_i$. This computation seems to be quite intensive, since it needs to track every single transition. Actually experiments showed that losing some transitions, which could happen in a gossip-based algorithm, does not have an important impact on results.

- The disconnection rate $\hat{\mu}$ using the relation:

$$\hat{\gamma_1} = \frac{\#Disconnections}{\Delta T}$$

and Little's law as in eq. 2:

$$\hat{\mu} = \frac{\hat{\gamma_1}}{\hat{n}} \tag{5}$$

- The abandon rate $\hat{P}$ can be computed in two equivalent ways. The first one is:

$$\hat{P} = 1 - \frac{\#Reconnections}{\#Disconnections}$$

while the second, which is the one used in our implementation, relies on the first balance equation in eq. (1) and it is:

$$\hat{P} = \frac{R}{\hat{\gamma_1}} \tag{6}$$

3.3 The Controller

The controller receives the estimations $\hat{\mu}$ and $\hat{P}$ and the target number of available fragments n' and uses eqs. (5) and (6) to compute the repair rate R as:

$$\boxed{R = \hat{\mu}\hat{P}n'} \tag{7}$$

Table 1 summarizes the symbols used.

symbol	meaning
μ	disconnection rate
λ	reconnection rate
P	abandon probability
R	repair rate
Q_1	queue of available fragments
Q_2	queue of temporarily unavailable fragments
γ_i	departure/arrival rate for queue Q_i
n	number of available fragments
m	number of temporarily unavailable fragments
n'	target number of available fragments
ΔT	estimation/update period

Table 1: Table of symbols.

4. IMPACT OF ESTIMATION TIME

The estimation time ΔT, which corresponds also to the update time of the repair rate R, is the most crucial parameter of our model. In this section we discuss the tuning of this parameter and explain its implications.

4.1 Impact on Bandwidth Usage

Proactive schemes work well in static environments in which there exist constant statistical properties, i.e. properties that once estimated never change. In such a case, the ideal choice would be to perform a preliminary *offline* estimation and then select an infinite ΔT. Any different choice would make the controller follow short-term fluctuations, doing unnecessary work on the one hand and unevenly using the bandwidth resources on the other.

Reactive schemes follow instantaneously any fluctuation of the system in the belief that no properties describing the long-term behavior of the system can be found. This, in principle, corresponds to setting $\Delta T = 0$, in which case the controller promptly reacts to any change. An example of such a condition is when we expect from the system massive correlated failures, where, without any possible prediction, most of available fragments suddenly disappear.

Our assumption is that while real systems may lack long-term statistical properties, they may have short-term properties that can be still exploited to make smoother the use of the bandwidth keeping providing data durability.

If we consider a system in which the model parameters change continuously at a given rate, our challenge is to choose the maximum ΔT that divides the time in segments in which the system can be approximated as being statistically stable.

This ideal choice would use the repair bandwidth in the smoothest possible way. In this case the residual fluctuations in the repair bandwidth are those strictly necessary to provide durability. If these fluctuations are not supported by the system, it just means that the available resources are not enough for what we are trying to achieve.

At this point, reactive schemes appear as the safest and the most conservative choice: they overshoot the instantaneous repair rate, but requiring, in certain periods, an excessive amount of the repair bandwidth. Reactive systems are very popular, because without any complex tuning they provide availability, but they pay a high cost in terms of a bursty resource consumption. Purely proactive schemes, instead, found very little attention in literature, because they work only for theoretical cases.

Tempo [4] is a very interesting work in this domain. The authors state that estimation is unreliable and instead introduce a per-node bandwidth budget. Using this budget and the available storage space, *Tempo* produces as many fragments as it can. This, of course, is a valid solution, but the right choice of the bandwidth budget is crucial since a wrong choice may waste repair resources or compromise the durability of the stored objects.

4.2 Robustness of the Estimation

The convergence speed is a key issue in a statistical estimator, and in our case puts a lower limit on ΔT.

If one tries to push system reactivity too much, it would degenerate to a situation in which estimation

does not make sense and a pure reactive scheme is more reliable. Besides, stating that the system has short-time statistical properties with a very high variation frequency, which is the condition that would require a very small ΔT, is equivalent to saying that statistical properties do not exist at all.

Furthermore the time needed to estimate the parameters depends on the parameters themselves. This means that in a dynamic environment a fixed choice of ΔT does not make sense. For this reason, in our implementation we do not fix ΔT, but D, the **average number of disconnections observed during an estimation period**: we use the last estimations of $\hat{\mu}$ and $\hat{n}$ to predict the time ΔT_D we expect to wait to observe D disconnections, with ΔT_D defined as:

$$\Delta T_D \triangleq \frac{D}{\hat{\gamma}_1} = \frac{D}{\hat{\mu}\hat{n}} \tag{8}$$

5. A HYBRID SCHEME FOR AVAILABILITY

Let us consider the control rule in eq. (7) using the real parameters and under the hypothesis in which the average number of available fragments is exactly n':

$$R = \hat{\mu}\hat{P}n' \Rightarrow R = P(\mu\overline{n}) = P\gamma_1 \tag{9}$$

Eq. 9 means that the objective of the controller is to make the repair rate equal to the rate of permanent failures, which corresponds to an **oracle system** able to tell apart the permanent departures from the transient ones. This ability, however, provides only durability, but it cannot give any guarantee on the availability. Indeed there might be periods in which a lot of peers are temporarily disconnected and some objects might not have enough available fragments to be reconstructed, while their existence in the system in the long run is not jeopardized.

A common pitfall is to consider durability and availability as completely separate objectives that can be achieved independently. The problem resides in the fact that the repair operation needs k fragments, i.e. *we need availability to assure durability*. Therefore, even an oracle may run into situations in which it cannot respond to a permanent failure because there are not enough fragments.

The conclusion is that durability can be provided without availability only in a statistical sense, as shown in [14]. However no deterministic guarantees can be provided.

As previously said our system tries to achieve durability, maximizing at the same time the likelihood of availability. To always guarantee availability, we propose a hybrid scheme in which the system switches to a purely reactive policy any time the number of available fragments hits a lower threshold TH_{pro}. During these *reactive* periods, which should be exceptional, the number of available fragments does not decrease as long as the system can keep up with all the disconnections, introducing, as a side effect, spikes in the bandwidth usage. If TH_{pro} is properly chosen, these spikes are again strictly needed and our approach still remains the best trade-off.

6. SYSTEM VALIDATION

To validate the system we designed and implemented an event-driven simulator. It simulates a simplified version of a distributed storage system on a set of peers whose behavior is described by availability traces it is fed with.

For these experiments, we created synthetic traces of the peer behavior using exponentially distributed disconnection and reconnection times. This assumption is very useful to validate the system. Indeed, if the model is correct we will obtain a distribution of the number of available fragments matching the expected theoretical one in eq. (3). Note, however, that, since the estimation is independent of the distribution used, a choice of a different distribution would not affect the results.

6.1 Model Validation

To validate the model we observe its behavior when the parameters are known. We choose a set of fixed parameters: $\mu = 1$, $P = 0.5$ and $\lambda = 2$ and the repair rate $R = 100 \cdot \mu \cdot P = 50$ which should provide, using eq. (2), an average number of available fragments $\overline{n} = 100$.

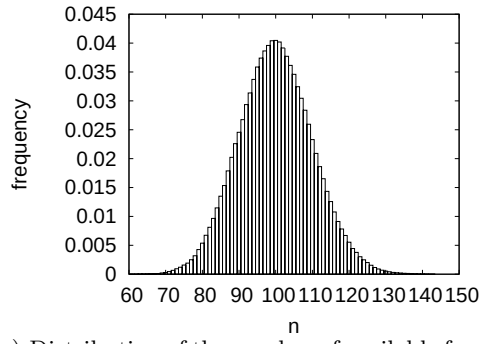
Figure 4(a) shows the distribution of the number of online fragments, which clearly fits a Normal distribution as we expected. Figure 4(b) instead depicts the instantaneous repair rate R_{inst}, which reflects roughly the bandwidth consumption due to the maintenance process. This measure is computed as the **inverse of the time elapsed between two consecutive repair events**:

$$R_{inst} = \frac{1}{t_{R_i} - t_{R_{i-1}}}$$

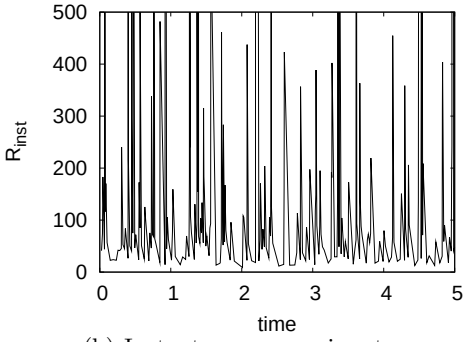
Since repair times are exponentially distributed, the repair events are not equally spaced, producing a high variability in the repair rate.[1]. This high variability is exactly what we want to avoid and suggests that the use of exponentially distributed repair times does not fit our needs.

We run a second experiment in which we use constant repair times, which violates the assumptions made in the model about exponentially distributed arrival rates. Figure 5 shows that this has practically no impact on the distribution of the available fragments that keeps fitting a Normal distribution and seems to be even nar-

[1]For graphical reasons we limited the y-range to 500. Rate spikes did attain values up to 50000 repairs per time unit.



(a) Distribution of the number of available fragments.



(b) Instantaneous repair rate.

Figure 4: Simulation with synthetic data, fixed parameters and exponentially distributed repair times.

rower. Obviously, in this case, the instantaneous repair rate is completely smoothed, i.e. constant.

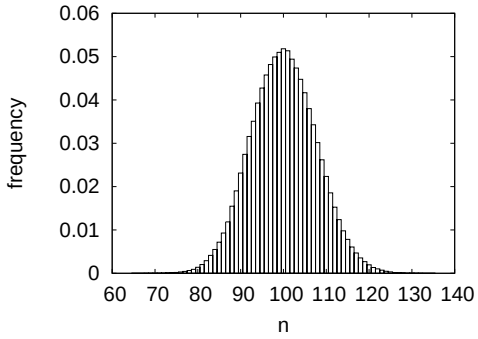
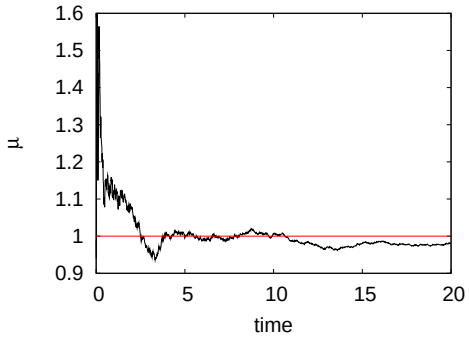


Figure 5: Simulation with synthetic data, fixed parameters and *constant repair times*. Distribution of the number of available fragments.

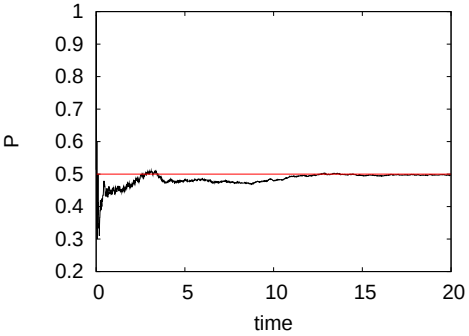
This synthetic scenario does not require any estimation since the parameters are known to be stable. In practice their values could have been obtained with an *offline* observation of the system behavior.

6.2 Estimator Validation

In this section we show the efficacy of the estimator. We run several experiments with different *fixed* values of the parameters and we observe the convergence of the estimation. For space limitations, in figure 6 we show only the results of a single case, where $\mu = 1$ and $P = 0.5$. The estimator is able to converge in about 5 time units. This convergence time is roughly proportional to γ_1, i.e. the number of samples (disconnections) observed per time unit, which is in turn proportional to μ.



(a) Estimation of the disconnection rate μ.



(b) Estimation of the abandon probability P.

Figure 6: Estimations with fixed parameters: $\mu = 1$ and $P = 0.5$.

All the experiments we ran pointed out two different issues that we already discussed in section 4.2:

1. Convergence of the estimation is not immediate. Even with fixed values it takes some time to obtain reasonable estimates. When we select a very small ΔT, we increase the reactivity of the system, but we are not able to infer its statistical properties.

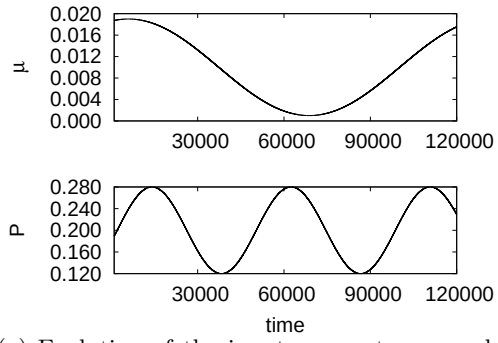
2. The convergence time depends on μ. This leads us to say that in a changing environment we cannot use a constant estimation period, but instead ΔT_D should be adapted to the order of magnitude of the parameter μ as we did in eq. (8).

6.3 Controller Validation

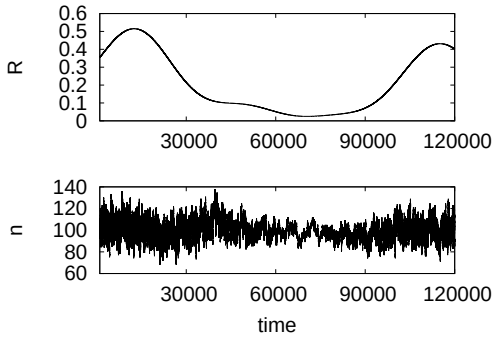
In this section we run experiments with varying parameters shown in figure 7(a) and with an ideal estimator that knows these parameters. This simulation aims

to show that all the considerations made about a system with fixed parameters are still valid in a dynamic environment.

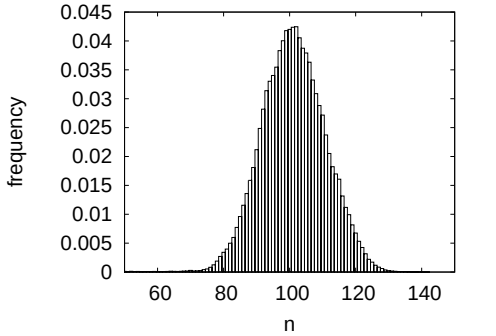
Results are shown in figure 7(b) and demonstrate clearly that the controller is able to maintain correctly the number of fragments. Moreover, the distribution of the number of available fragments in figure 7(c) still resembles a Gaussian.



(a) Evolution of the input parameters μ and P.



(b) Repair rate R and evolution of the number of available fragments.



(c) Distribution of the number of available fragments.

Figure 7: Controller Validation: simulation with varying parameters and ideal estimator

7. EXPERIMENTS

We are mainly interested in two aspects, the capacity to assure durability and the smoothness of the instantaneous repair rate. The durability can be easily evaluated looking at the distribution of the number of available fragments n.

Assessing the smoothness of the instantaneous repair rate is a bit more complex. Note that the ideal case is not necessarily the one in which the instantaneous repair rate is *constant*, but the one in which its variations are minimal given the variations in the system. This minimum corresponds to the ideal instantaneous repair rate, which is the rate we would select if we were able to know instantaneously the exact system parameters, as we did in section 6.3.

Formally speaking, the **ideal instantaneous repair rate** is a continuous signal $R_{ideal}(t)$ and is given by the following relation:

$$R_{ideal}(t) \triangleq \mu(t)P(t)n'$$

For every repair event R_i we compute $R_{diff}(t_{R_i})$, which is the instantaneous repair rate we observe and the ideal instantaneous repair rate in that instant:

$$R_{diff}(t_{R_i}) \triangleq R_{inst}(t_{R_i}) - R_{ideal}(t_{R_i}) \qquad (10)$$

The discrete sequence $R_{diff}(t_{R_i})$ measures how far is the instantaneous repair rate from the ideal one. To characterize this measure we use its standard deviation. The closer $std(R_{diff})$ is to zero, the closer our system is to the ideal case.

7.1 Evaluation with Synthetic Data

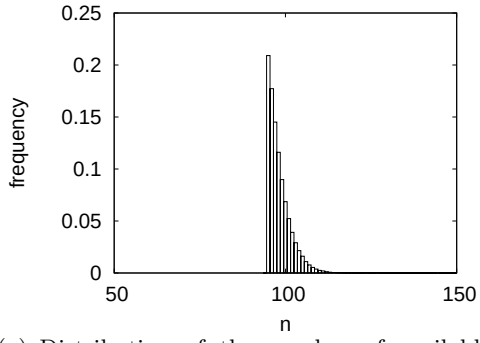
In these experiments we use synthetic traces, again with the assumption of exponentially distributed disconnection and reconnection times. The objective is to show that our approach obtains a higher smoothness than a reactive scheme and to evaluate the impact of the parameter D on the smoothness.

We choose the same sine waves for μ and P as in figure 7(a) . The duration is 300000 time units and we choose $n' = 100$, which is a good value to show all the dynamics of our system.

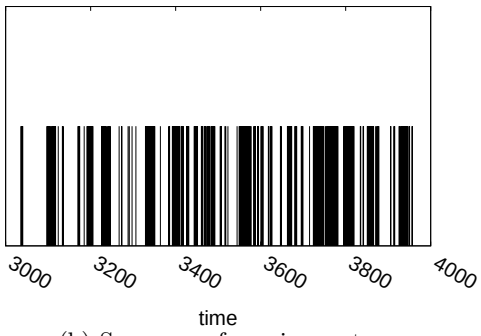
The reactivity is controlled through the parameter D, which in turn influences ΔT_D. Since for too small values of D the estimation is not reliable and for too big values the distribution of the number of available fragments degrades too much, we choose for D values between 50 and 2000.

The basis for our comparison is a generic reactive scheme, which performs a single repair whenever the number of available fragments is below a threshold TH_{reac}. In these experiments, we choose the threshold $TH_{reac} = 95$, which is the value that produces an average number of online fragments equal to the one provided by our proactive scheme, namely $n' = 100$. In figure 8(a) we plot the distribution of the number of available fragments, which represents an extremely good result. However, the cost of the reactive scheme is a bursty repair

activity shown in figure 8(b), where the sequence of repair events is depicted.



(a) Distribution of the number of available fragments.



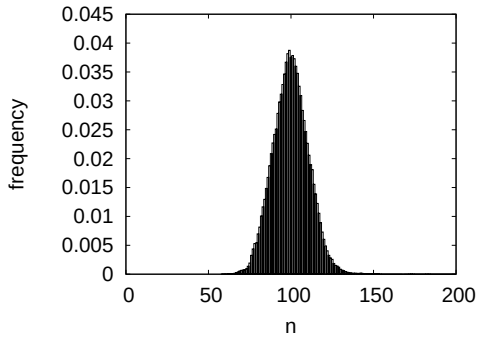
(b) Sequence of repair events.

Figure 8: Reactive Scheme: simulation with synthetic data.

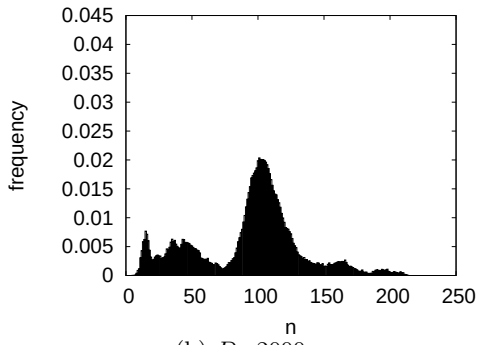
The objective of our scheme is to equally space the repair events, while still assuring a reasonable distribution of the number of available fragments, which tends to get worse when D is increased (Compare in figure 9 the case of $D = 50$ and $D = 2000$).

The results obtained with $D = 2000$ are clearly not acceptable because of the low level of availability. This is due to the fact that D is too big with respect to the parameter dynamics and the estimator is not able to cope with their changes. This is clearly shown in figure 10(b), where the instantaneous repair rate $R_{inst}(t_{R_i})$ is compared with the ideal one $R_{ideal}(t)$. Comparing figures 10(a) and 10(b) we also see that smoothness of the rate is strongly related with D.

The aggregate results for all the values of D are shown in figure 11, where the mean, the 5-percentile and the 95-percentile of the number of available fragments are shown in 11(a) and the standard deviation of R_{diff}, as defined in eq. (10), is shown in 11(b). Note that at $D = 0$ we associated the results for the reactive scheme. These results give a clear picture of the trade-off in the choice of D, namely that the increased smoothness has a cost in terms of the distribution of the number of available fragments.



(a) D=50.



(b) D=2000.

Figure 9: Simulation with synthetic data: distribution of the number of available fragments for $D = 50$ and $D = 2000$.

The poor availability in case of D being too big motivates the need for a hybrid scheme that puts a lower-bound on the number of available fragments, even when D is not properly chosen.

Experiments with the Hybrid Approach

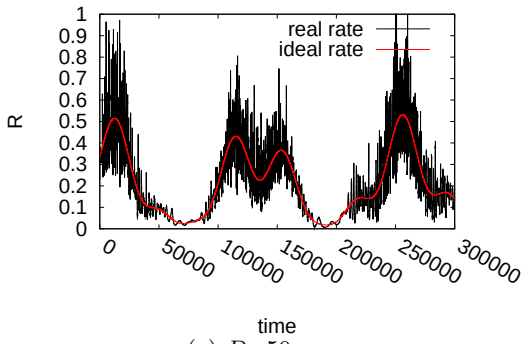
In these experiments we evaluate the hybrid approach presented in section 5, and set the threshold $TH_{pro} = 50$.

The essence of such approach is shown in figure 12, where the distribution of the number of available fragments for $D = 2000$ is plotted. The number of fragments, thanks to the hybrid scheme, never goes below the threshold of 50.

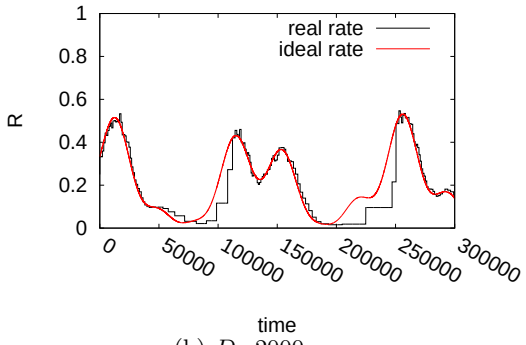
The aggregate results are not dissimilar from the ones obtained without the Hybrid Approach.

7.2 Evaluation with PlanetLab Traces

We tested our scheme using real availability traces from PlanetLab All Pairs Ping [17]. These traces consist in the availability status of 669 nodes and were obtained by means of pings sent every 15 minutes between all pairs of the concerned PlanetLab nodes, starting from January 2004 for about 500 days. These traces are publically available at [9]. We used data from the file `pl-app-cleaned.avt`.



(a) D=50.



(b) D=2000.

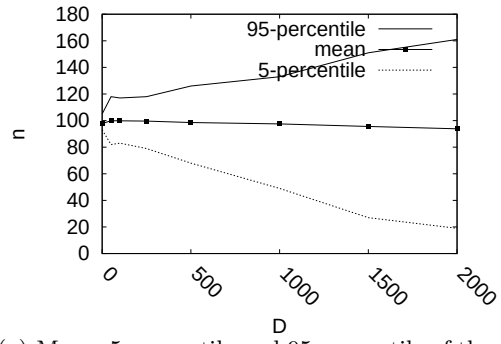
Figure 10: Simulation with synthetic data: comparison of R_{inst} and R_{ideal} for $D = 50$ and $D = 2000$.

We run and compare three different schemes:

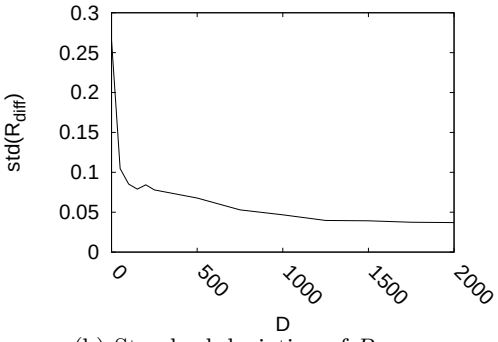
- Our proactive scheme with the hybrid approach, a threshold of $TH_{pro} = 50$ and a target number of available fragments of $n' = 100$.
- A reactive scheme with a threshold $TH_{reac} = 80$. We choose experimentally the value of 80 that provides an average number of available fragments equal to the one provided by our proactive scheme ($n' = 100$).
- An oracle scheme which also starts with the same initial number of fragments as other schemes and represents a system that is able to distinguish transient from permanent failures, triggering a repair only in case of a permanent failure.

Since we do not know what are the *real* parameters of the system, we can neither compute R_{ideal} nor R_{diff}. To evaluate the smoothness of the repair rate in this case we use directly the cumulative amount of repairs done over time. This curve, already used in [4], gives us the total amount of work done by the different algorithms and its derivative expresses the instantaneous repair rate at which this work was done.

To easily compare the schemes we selected two values of D: $D = 500$ and $D = 1000$.



(a) Mean, 5-percentile and 95-percentile of the number of available fragments.



(b) Standard deviation of R_{diff}.

Figure 11: Simulation with synthetic data: reactivity vs fragments availability and rate smoothness.

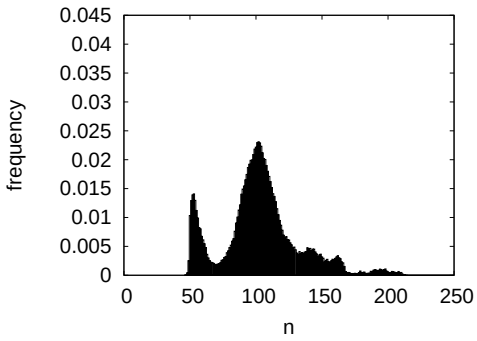
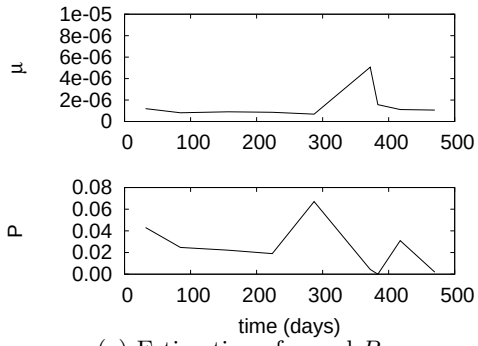


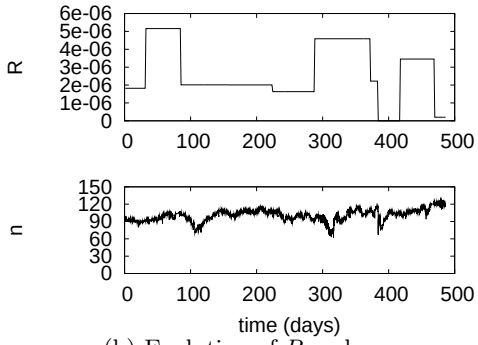
Figure 12: Hybrid Approach: Simulation with synthetic data. Distribution of the number of available fragments for $D = 2000$ and $TH_{pro} = 50$.

In figure 13 we show the time evolution of our proactive system for $D = 500$, where the estimations of μ and P are shown in 13(a), while the repair rate selected and the evolution of the available fragments obtained are shown in 13(b).

In figure 14(a) the distribution of the available fragments with the three schemes is shown, while in 14(b) the cost of the repair process is shown in terms of cumulative number of repairs performed.

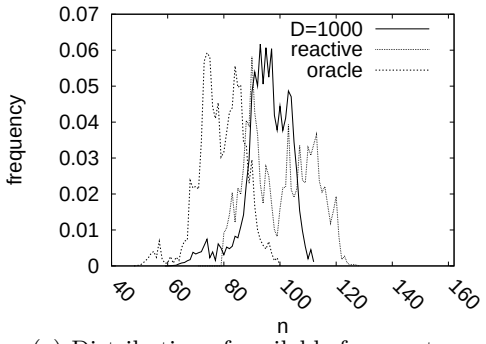


(a) Estimation of μ and P.

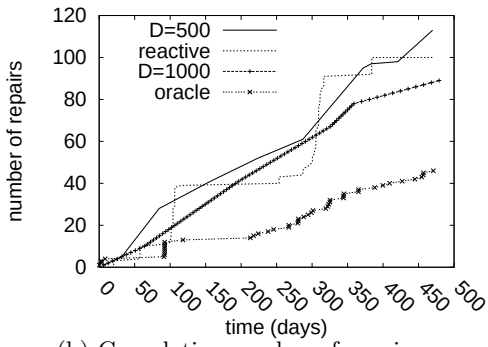


(b) Evolution of R and n.

Figure 13: Proactive scheme with $D = 500$ on PlanetLab traces.



(a) Distribution of available fragments.



(b) Cumulative number of repairs.

Figure 14: Simulations with PlanetLab traces.

The oracle, which is *a scheme that is practically not achievable*, requires the lowest number of repairs, since it creates new fragments only when peers permanently fail. Although an oracle system would not perform unnecessary work, its distribution of the number of available fragments is shifted towards lower values of n running into the risk of being below the level required to assure availability.

The reactive scheme and our proactive scheme have a distribution of the number of available fragments with the same mean value, namely 100; the difference resides in the fact that in the reactive scheme the threshold mechanism prevents the number of available fragments to fall below 80, while in the proactive schemes we tolerate lower values. This situation is still acceptable and is compensated by the advantage of having a much smother repair rate as shown in 14(b). While the total number of repairs is comparable, the curves in 14(b) show that the reactive scheme is more bursty than the proactive schemes. Moreover, as expected, a large D produces a better smoothness.

The differences in the dynamics of the two schemes may be understood looking at day 300, where a lot of transient failures took place. To face such an event, a reactive scheme simply performs a lot of repairs producing a step in the curve and a spike in the resource consumption. The proactive schemes, instead, up to that moment have already done a higher number of repairs, i.e. they have done part of the work in advance, and can absorb the massive disconnection without negative impact on the repair process. At the next parameter update, this higher disconnection rate is taken into account slightly increasing the repair rate. If the disconnection peak was bigger and was not sustainable by the chosen rate, the number of available fragments would have fallen below the threshold TH_{pro} triggering the hybrid scheme to initiate a *reactive period* in response of the excessive churn.

In general, proactive schemes tend to have a higher number of repairs. This, as already suggested, is due to the fact that proactive schemes work in *advance* to spread the repairs over time. *This anticipation of repairs is the only way to smooth the rate without compromising the durability.* The price to pay, however, is that part of the fragments created in advance may be lost, because of permanent failures, before they are needed [3]. The higher the abandon probability P the more pronounced this effect will be.

8. CONCLUSION AND FUTURE WORK

We proposed a novel framework for managing redundancy based on the estimation of the peer behavior.

Our system combines the resilience of reactive schemes with the smoothness of proactive schemes. This can be

considered as a general approach, in which the duality reactive or proactive becomes a specific case of a wider approach tunable with respect to the dynamics of the failure behavior.

We validated the proposed scheme and demonstrated its effectiveness using synthetic data and availability traces of PlanetLab nodes.

We also clarified the difference between availability and durability and their relationship, which is often misunderstood.

We see possible extensions of our scheme in the following two directions.

Automatic tuning of the parameter D. Even if the ability of the system to provide durability is not very sensitive to the choice of D and we chose the working range manually, it would be nice to find a way to automatically tune its value. Our objective is to build a system that automatically finds a good tradeoff between reactivity and rate smoothness and selects the optimal value of D, even if the starting one is not optimally chosen or if the system evolves in time.

Hybrid redundancy scheme. It is well known that for a given amount of redundancy, parity encoding offers a higher degree of reliability than replications. However, as has been pointed out previously [16, 20], to regenerate one lost fragment, parity encoding schemes need to read k fragments. This means that for parity encoding schemes the volume of the I/O and network traffic will be k times higher than the amount of regenerated fragments. Depending on the rate R of permanent node failures, parity encoding schemes may result in a very high amount of traffic. For this reason, a **hybrid redundancy scheme** that combines replication to keep the repair traffic low and parity to protect against massive (correlated) node failures seems very attractive (see e.g. [10]). We plan to study how our proactive scheme can be extended to hybrid redundancy schemes.

Acknowledgments

The first author is supported by a PhD Scholarship from Microsoft Research.

9. REFERENCES

[1] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available and reliable storage for an incompletely trusted environment. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

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

[3] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Symposum on Networked Systems Design and Implementation (NSDI)*, 2006.

[4] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, J. K. M. Frans Kaashoek, and R. Morris. Proactive replication for data durability. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[5] F. Dabek, K. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Symposium on Operating Systems Principles (SOSP)*, 2001.

[6] A. Datta and K. Aberer. Internet-scale storage systems under churn - a study of the steady state using markov models. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2006.

[7] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

[8] C. Fragouli, J.-Y. L. Boudec, and J. Widmer. Network coding: An instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.

[9] B. Godfrey. Repository of availability traces. http://www.cs.berkeley.edu/ pbg/availability/, 2006.

[10] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Symposum on Networked Systems Design and Implementation (NSDI)*, 2005.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer System*, 23(3):219–252, August 2005.

[12] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[13] J. Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[14] G. Lefebvre and M. J. Feeley. Separating durability and availability in selfmanaged storage. In *ACM SIGOPS European Workshop (SIGOPSEW)*, 2004.

[15] S. Rhea et al. OpenDHT: A public DHT service and its uses. In *SIGCOMM*, 2005.

[16] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs.replication. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[17] J. Stribling. Planetlab all pairs ping. http://infospect.planet-lab.org/pings.

[18] K. Tati and G. M. Voelker. On object maintenance in peer-to-peer systems. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[19] K. S. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*. John Wiley & Sons, 2nd edition, 2001.

[20] G. Utard and A. Vernois. Data durability in peer to peer storage systems. In *IEEE International Symposium on Cluster Computing and the Grid*, 2004.