Probabilistic Failure Detection for Efficient Distributed Storage Maintenance[*]

Jing Tian[*], Zhi Yang[*], Wei Chen[†], Ben Y. Zhao[‡], Yafei Dai[*]
[*]*State Key Lab for Adv Opt Comm. Syst & Networks, Peking University, Beijing, China*
[†]*Microsoft Research Asia, Beijing, China*
[‡]*Computer Science Department, UC Santa Barbara, CA, USA*
jing.tian@gmail.com; {yangzhi,dyf}@net.pku.edu.cn;
weic@microsoft.com; ravenben@cs.ucsb.edu

Abstract

Distributed storage systems often use data replication to mask failures and guarantee high data availability. Node failures can be transient or permanent. While the system must generate new replicas to replace replica lost to permanent failures, it can save significant replication costs by not replicating following transient faults. Given the unpredictability of network dynamics, however, distinguishing permanent and transient failures is extremely difficult. Traditional timeout approaches are difficult to tune and can introduce unnecessary replication.

In this paper, we propose Protector, an algorithm that addresses this problem using network-wide statistical prediction. Our algorithm drastically improves prediction accuracy by making predictions across aggregate replica groups instead of single nodes. These estimates of the number of "live replicas" can guide efficient data replication policies. We prove that given data on node down times and the probability of permanent failures, the estimate given by our algorithm is more accurate than all alternatives. We describe two ways to obtain the failure probability function driven by models or traces. We conduct extensive simulations based both on synthetic and real traces, and show that Protector closely approximates the performance of a perfect "oracle" failure detector, while significantly outperforming timeout-based detectors using a wide range of parameters.

1. Introduction

Distributed storage is an important class of distributed systems of which we have seen an increasing number of applications in various settings, such as large data centers supporting internet services (*e.g.* Google File System [1]) and hosted storage services (*e.g.* Amazon's S3 [2] and Google's GDrive [3]), and wide-area peer-to-peer systems (*e.g.* Oceanstore [4]). One of the key issues to address in distributed storage is how to tolerate failures and departures of nodes in the system (for simplicity, from now on we use failures to refer to both failures and departures of nodes in the systems). To meet the service level agreements (SLAs), a system must achieve certain levels of availability for the data stored in the system. To do so, a simple and effective approach adopted by most current distributed storage systems is data replication [4-11]. Each data object is replicated *m* times so that even if some replicas become unavailable due to their host failures, the data object is still available by accessing other online replicas. As nodes enter and exit the network, the storage system maintains data availability by monitoring and proactively maintaining a number of replicas for each data object.

Maintaining data availability incurs a heavy cost on storage systems, the biggest of which is the bandwidth required to generate additional object replicas. Existing work [12] has shown that frequent data replication incurs high bandwidth costs that may congest the network and cripple the entire storage system. However, reducing the number of replicas maintained might result in an unacceptable level of data availability. Therefore, the challenge in building today's storage systems is to carefully navigate this efficiency-availability tradeoff, by reducing replication costs as much as possible while maintaining the desired level of data availability.

This challenge is further complicated by the fact that node failures can be categorized as permanent or transient failures. A failed node will not recover following a permanent failure, and thus object replicas stored on the node are permanently lost. In contrast, a node suffering a transient failure eventually recovers and rejoins the network, bringing back its stored replicas. Clearly, permanent failures must be addressed by restoring the level of data replication after a node is permanently

[*] This work is supported by National Grand Fundamental Research 973 program of China under Grant No.2004CB318204, China 863 project under grant No.2007AA01Z100

failed. This requires the system to generate new replicas on other online nodes. However, a system with sufficient replicas can tolerate some transient failures without sacrificing data availability. Therefore, it is desirable that an efficient storage system would accurately distinguish between permanent and transient failures, and only pay the costs of object replication following permanent failures.

Distinguishing permanent and transient failures turns out to be a difficult task that requires accurate prediction of individual machine behavior. While not explicitly addressed by early systems, recent research work generally adopts one of two approaches: timeout thresholds [4, 7, 10, 11] and lazy replication [5, 6, 13]. In the timeout threshold approach, nodes wait for some preset timeout period before identifying a failure as permanent and acting to restore replication levels. While simple, this approach is error-prone, and choosing the timeout parameter walks a fine line between high values that produce false negatives and reduce availability, and low values that produce false positives and unnecessary data replication. In the lazy replication approach, the system generates extra replicas to postpone data recovery so that many transient failures can be masked by extra replicas. However, this approach incurs significant replication costs and still requires the use of a timeout parameter.

The key insight in our work is that we recognize that failure detection is difficult on a per-node basis, and propose instead to predict data availability across the entire group of replicas. By looking at the entire replica group, *i.e.* all nodes hosting replicas of the same object, we avoid the difficult task of detecting a single transient failure. Instead, we use a probabilistic method to estimate the total number of *remaining replicas* that are still in the system (*i.e.* replicas residing on active nodes or nodes experiencing transient failures). The first step of our method is to obtain a failure probability function that represents the probability that a node has permanently failed given an observed failure of d time units. We propose two ways to obtain this probability function -- using a simple Markov failure model, or extraction from historical traces of node failure events. Next, we aggregate failure probability functions across all nodes in the replica group to derive an estimate of the number of remaining replicas in the group. An important result here is that the estimate obtained by this method is the best among all estimation methods including timeout-based ones (under a measure called accuracy rate). Finally, we use this availability estimator to determine whether data recovery is necessary to reach the desired replica target. We call this prediction scheme PRObabilistic deTECTOR (or Protector). Intuitively, Protector uses probabilistic estimates on the entire replica group to reduce prediction errors, balancing false positives for some nodes against false negatives for others.

We make three key contributions in this paper. First, we describe in detail the Protector algorithm for probabilistic group-based availability estimation (Section 2). Second, we prove that among all techniques based on node downtimes (including timeout-based failure detectors), Protector achieves the best accuracy rate in estimating the number of remaining replicas (Section 2.3). Finally, we compare the effectiveness of Protector against timeout-based techniques through extensive simulations using both synthetic traces generated by a Markov failure model and actual traces collected from a large-scale peer-to-peer storage system. Our results show that not only does Protector outperform alternative approaches for most timeout parameters, it also approximates the performance of the ideal oracle failure predictor (Section 3). As we discuss in Section 4, Protector can complement the extra replica approach to provide better protection against transient failures while reducing the extra cost associated with maintaining the extra replicas.

2. Probabilistic Failure Detection

In this section, we describe in detail the Protector probabilistic replica maintenance approach, and analyze its effectiveness. We begin by first describing the high level replica maintenance problem as context. We then describe the Protector approach in detail, and prove that its estimate on the number of remaining replicas is the most accurate. Finally, we describe in detail how to derive Protector's failure probability function through both a Markov failure model and extraction from measurement traces of failure events.

2.1. Efficient Data Recovery

Protector is designed to guide the replication decisions of a reliable distributed storage system. The goal of the replication layer is to maintain a target replication level for each data object while minimizing the replication bandwidth overhead. The replication target t_r can be determined as a function of the desired level of data availability (as in [6, 9, 13, 14]). One simple way to obtain t_r is as follows. Suppose the target availability is $p_o \in (0, 1)$, and the availability of each individual host node (before the host fails permanently) is p_c. The availability of a data object with x replicas, which is the probability that at least one of the x replicas of the object is available, is $1-(1-p_c)^x$ (assuming failures of each individual nodes are independent). Thus, to satisfy target availability p_o, the target number t_r of replicas is the minimum x making $1-(1-p_c)^x \geq p_o$, which means:

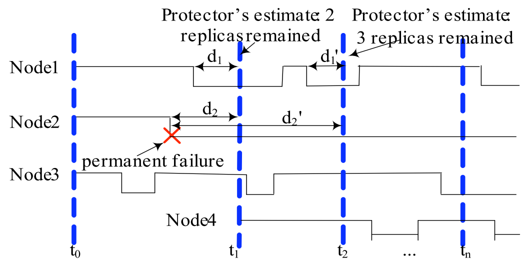
Protector's estimate: 2 replicas remained
Protector's estimate: 3 replicas remained
Node1
Node2
Node3
Node4
d1
d1'
d2
d2'
permanent failure
t0
t1
t2
...
tn

Figure 1. Data recovery in Protector ($t_r = 3$).

$$t_r = \left\lceil \frac{\log(1-p_o)}{\log(1-p_c)} \right\rceil \tag{1}$$

For each object, given a global target replication level t_r, the replication layer periodically (*e.g.* every one hour) computes an estimate m of the current number of replicas remaining in the system. The estimate m is computed using the Protector algorithm described later in this section. If $m < t_r$, then the replication layer generates $t_r - m$ new replicas randomly among online nodes in the system; if $m \geq t_r$, no actions are necessary. Figure 1 shows a simple example with $t_r = 3$. At time t_1, only node 3 is up and Protector estimates that there are two replicas remaining in the system, and thus the system generates a new replica on node 4. Later at time t_2, nodes 1 and 2 are down, nodes 3 and 4 are up, and Protector estimates that there are 3 remaining replicas (*i.e.* a replica on either node 1 or 2 is not permanently lost and will become available later). Therefore no additional replication is necessary.

Our basic scheme is suitable for immutable storage systems such as block storage systems, where objects will not be updated. It is also possible to incorporate Protector into mutable storage systems with strong consistency requirements. In such systems, an update log is typically maintained in parallel with the data object so that when a replica is recovered from a transient failure, the missing updates are applied on the replica instead of regenerating a new replica entirely. In this case, Protector can still be used in the maintenance of data replicas while a separate replication mechanism for strong consistency is used for update log replications.

In this paper, we focus on the detection and data recovery algorithm. In actual implementations, the algorithm can be implemented either at a master node in a centrally controlled storage system such as the Google file system [1], or in a distributed manner among all replica groups in decentralized storage systems such as DHT-based peer-to-peer storage [4, 6, 11].

2.2. Probabilistic Estimation in Protector

Protector is a probabilistic availability estimation algorithm that improves its prediction accuracy by aggregating its prediction across the entire replica group (*i.e.* all nodes hosting replicas for a given object). Consider an object with the target replication level t_r. Suppose that at the time of the estimate, there are n total nodes hosting replicas of the object, a subset of which are online. Let the n nodes be numbered from 1, 2, … to node n. Let d_i be the down time of node i, and $d_i = 0$ if node i is up at the time (*e.g.* d_1 and d_2 in Figure 1 for nodes 1 and 2 for the estimate at time t_1). The down time of a node begins when a low-level failure detector declares a failure of a node based on message delay patterns. Such failure detectors have been extensively studied (*e.g.* [15-18]).

Let TTR_i be the random variable representing "time-to-recover" of node i, that is, the time from when node i fails to the time when node i recovers. TTR_i could take value ∞, which means node i failed permanently and will not recover. When $d_i > 0$, we define function $F_i(d_i) = \Pr(TTR_i = \infty \mid TTR_i \geq d_i)$. That is, $F_i(d_i)$ is the probability that node i permanently fails given that it has already been offline for d_i time units. Function $F_i(d_i)$ is the key function in Protector, and we show two approaches to derive it later in this section. For now, we assume that $F_i(d_i)$ is known for any $d_i > 0$. We define $F_i(0)$ to 0, which means if node i is online, it cannot be a permanent failure.

Let X be a random variable representing the number of remaining replicas in the system at the time of estimation. Given $F_i(d_i)$ for all node $i = 1, 2, \ldots, n$, we can compute the probability of $X = k$ as follows.

$$\Pr(X = k) = \sum_{\substack{S \subseteq \{1,2,\ldots,n\} \\ |S| = k}} \prod_{i \in S} (1 - F_i(d_i)) \prod_{j \notin S} F_j(d_j) \tag{2}$$

The interpretation of the formula is as follows: Given any set S with size k, the first product in the formula is the probability that all nodes in S are online or having transient failures, while the second product is the probability that all nodes not in S have permanent failures and thus their replicas are permanently lost. Thus, the probability that exactly k replicas remain in the system is the sum of probabilities that exactly nodes in S have remaining replicas over all possible sets S.

Finally, Protector picks the m that maximizes $\Pr(X = m)$, *i.e.* $\Pr(X = m) = \max\{\Pr(X = k) \mid k = 0, 1, 2, \ldots, n\}$, and returns m as the estimated number of remaining replicas in the system.

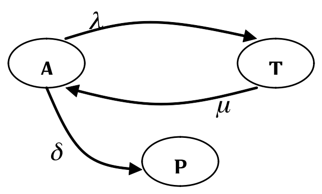

Figure 2. Markov failure model.

2.3. Proving Protector's Optimality

We now show that Protector's estimate is optimal in the measure *accuracy rate*, which is the probability that the estimate correctly matches the reality, among all estimation methods based on function $F_i(\cdot)$ and current downtime d_i. Ultimately, any availability estimation function must compute the number of replicas still alive in the system, which is then used to determine if additional replication is necessary. For example, approaches using timeout periods use a timeout threshold TO_i and mark a node i as permanently failed when $d_i > TO_i$. An accurate function $F_i(\cdot)$ may help the detector to derive a good timeout value, but in the end it uses timeouts to decide on the number of remaining replicas. Other methods may decide this number in a probabilistic way. In general, all these methods can be considered as giving an estimate of $X=k$ with a probability p_k, for $k=1,2,\ldots n$, such that $\sum_{k=1}^{n} p_k = 1$. Then the accuracy rate r is:

$$\begin{aligned} r &= \sum_{k=1}^{n} \Pr\{X = k\} p_k \\ &\le \sum_{k=1}^{n} \max_{i \in \{1,\cdots,n\}} \{\Pr\{X = i\}\} p_k \\ &= \max_{i \in \{1,\cdots,n\}} \{\Pr\{X = i\}\} \sum_{k=1}^{n} p_k \\ &= \max_{i \in \{1,\cdots,n\}} \{\Pr\{X = i\}\} \\ &= \Pr\{X = m\} \end{aligned}$$

Therefore, the accuracy rate of any estimate method cannot exceed $\Pr(X = m)$, which is what Protector achieves, assuming that Protector knows $F_i(\cdot)$. Hence, we have the following proposition.

Proposition: *Among all detectors with the knowledge of downtime d_i and function $F_i(\cdot)$ for all nodes 1 to n, Protector achieves the best accuracy rate in estimating the number of remaining replicas in the system.*

Note that an important implication is that, no matter how one tunes the timeout scheme to find the best timeout threshold, even customizing best timeout thresholds for different nodes, ultimately it is still used to obtain the number of remaining replicas (*i.e.* excluding the number of replicas on nodes to be judged as permanently failed by the timeout thresholds). Therefore, any timeout scheme is covered by the above proposition, and Protector performs as well or better than any timeout-based approach. Given the challenge of deriving accurate timeout values, Protector is likely to significantly outperform typical timeout schemes in practice.

Protector's method can be viewed as a maximum likelihood estimation, since its estimate is the most likely one that occurs in reality. Other estimation method, such as taking the mean, which minimizes mean square error, could also be explored as an alternative.

2.4. $F_i(d)$ using Markov Failure Model

.

The first method to derive function $F_i(d)$ is by a Markov failure model, which characterizes the failure and recovery behaviors of a node as a stochastic process with exponentially distributed transition periods between different states of the node (*i.e.* available, transiently-failed, or permanently-failed). The advantage of this approach is twofold. First, it allows simple calculation of $F_i(d)$ with only a few parameters. Second, it allows further analytical studies or Monte-Carlo simulations to evaluate the effectiveness of Protector and other methods on different failure environments (*e.g.* dynamic systems with frequent failures and recoveries or stable systems with only occasional transient failures and rare permanent failures), all under the same stochastic model. The disadvantage of this method is its reduced accuracy in reflecting actual system behaviors, which may not follow exponential distributions exactly. However, our prior work indicates that the exponential distribution is a reasonable approximation to failures and recoveries in distributed storage systems [19]. In the evaluation section, we will further compare this method with the next method, which uses historical system traces.

The Markov failure model of a node is a continuous-time Markov chain given in Figure 2. Each node has three states: A means available, T means transiently-failed, and P means permanently-failed. State A can be transitioned to T or P; T can be transitioned back to A; and P is a sink with no transition back to any other state. The durations of all transitions follow exponential distributions with transitions rates denoted on the edges of the transitions.

With the Markov model, function $F_i(d)$ is derived as

$$F_i(d) = \frac{\delta}{\delta + \lambda \cdot e^{-\mu d}} \qquad (3)$$

Its derivation is by the application of the Bayesian's Theorem as follows.

$$
\begin{aligned}
F_i(d) &= \Pr(TTR = \infty \mid TTR \geq d) \\
&= \frac{\Pr(TTR=\infty)\cdot\Pr(TTR\geq d \mid TTR=\infty)}{\Pr(TTR=\infty)\cdot\Pr(TTR\geq d\mid TTR=\infty)+\Pr(TTR\neq\infty)\cdot\Pr(TTR\geq d\mid TTR\neq\infty)} \\
&= \frac{\Pr(TTR=\infty)}{\Pr(TTR=\infty)+\Pr(TTR\neq\infty)\cdot e^{-\mu d}},
\end{aligned} \quad (4)
$$

where $\Pr(TTR = \infty)$ is the probability that a failure is a permanent failure. Since the failure transitions from state A has two competing transitions: one is to T with rate λ, and the other is to P with rate δ, it is easy to verify that

$$\Pr(TTR=\infty) = \frac{\delta}{\delta+\lambda} \quad (5)$$

Combining (4) and (5), we obtain equation (3).

From equation (3), we know that we can obtain $F_i(d)$ as long as we have the three rates, λ, μ, and δ related to failures and recoveries. They correspond respectively to the mean time to failure (*MTTF*) (given that the failure is transient), the mean time to recovery (*MTTR*), and the mean life time (*MLT*) of the node. In particular, $\lambda=1/MTTF$, $\mu=1/MTTR$, and $\delta=1/MLT$. Therefore, if we can obtain these parameters about node failures (*e.g.* from historical traces of node failures), we can easily compute $F_i(d)$ for Protector using equation (3).

2.5. Computing $F_i(d)$ using Failure History

In some cases, failures and recoveries may deviate from an exponential distribution and introduce errors into the Markov model approach. We describe here an alternative approach. When actual failure traces are available, Protector can derive a more accurate $F_i(d)$ directly from measurement trace.

First, based on application requirement and failure distribution, we need to determine how long a continuous down time period is considered a permanent failure. Second, based on the failure and recovery behaviors of the nodes, one can extract statistics for transient failures from past events. In particular, suppose that one monitors the failure and recovery behaviors of all nodes for a period of time (*e.g.* one month), and obtain the following statistics: M, the number of permanent failures among all nodes, and $N(d)$, the number of transient failures whose *TTR* is larger than d through the period (therefore $N(0)$ is the number of all transient failures). Thus, the probability that any failure is permanent is given by:

$$\Pr\{TTR=\infty\} = \frac{M}{N(0)+M} \quad (6)$$

and the probability that a failure's (permanent or transient) *TTR* is larger than d is given by:

$$\Pr\{TTR\geq d\} = \frac{M+N(d)}{N(0)+M} \quad (7)$$

Combining the above two equations we get the conditional probability $F_i(d)$ for any node i:

$$F_i(d) = \Pr\{TTR=\infty \mid TTR\geq d\} = \frac{\Pr\{TTR=\infty, TTR\geq d\}}{\Pr\{TTR\geq d\}} = \frac{\Pr\{TTR=\infty\}}{\Pr\{TTR\geq d\}} = \frac{M}{M+N(d)} \quad (8)$$

Using (8) the system can compute the $F_i(d)$ of node i. Note that the above computation does not assume exponential distributions on failures and recoveries, and thus equation (8) is applicable to general failure scenarios. The disadvantage of this approach is that it requires a large memory footprint to store and access $N(d)$ for a large number of d values, while the Markov model only needs three parameters (*MTTF*, *MTTR*, and *MLT*) to compute $F_i(d)$.

Using historical data for failure detection is commonly used for adaptive failure detectors (*e.g.* [16-18, 20]). Of course, system behavior may change over time, so actual implementation needs to use recent history to keep detection accuracy. One may also argue that given the historical traces, the timeout for a timeout-based failure detector can be tuned offline to achieve the best availability-cost tradeoff. While this could be true theoretically, we believe that Protector's approach is more efficient. With the Protector, we only need to obtain a few statistics (*MTTF*, *MTTR*, *MLT* for the first method and M, $N(d)$ for the second method), which can be easily computed by running through the trace. To tune a timeout, however, one may have to simulate a large amount of runs with different timeout values in order to achieve the best timeout value.

An important remark is now in order. In both methods, we need to identify permanent failures in the historical traces in order to extract the quantities used for our calculation (*MTTF*, *MTTR*, *MLT* in the first method and M in the second method). To do so, we need a threshold value such that failures lasting longer than the threshold are deemed as permanent failures (we use 30-days in our evaluation). We would like to emphasize that this threshold cannot be used as a timeout value in the timeout-based failure detector to determine permanent failures. This is because for a subsequent analysis of the historical trace, one would choose an extremely large threshold to determine the permanent failures exactly. Online predictions cannot afford such large timeouts, since it may put data availability in danger. Our evaluation results clearly show this distinction.

3. Evaluation

We evaluate the performance of Protector through simulations based on both a Markov failure model and actual system measurement traces. The two sets of simulations serve to validate different aspects of the advantages of Protector. The simulations based on the Markov failure model validate whether the optimality in estimating the number of remaining replicas in the system (shown in Section 2.3) leads to well-balanced tradeoff between cost and availability. The simulations based on actual system traces further validate whether Protector still provides good performance when the system behavior does not exactly follow the Markov model. Our results show that in both cases the cost and availability tradeoff provided by Protector is very close to that of an oracle detector that always knows exactly if a failure is permanent or not.

3.1. Evaluation via Markov Failure Model

We first use the Markov failure model described in Section 2.4 to evaluate Protector. All nodes in the system behave according to the Markov failure model given in Figure 2. With this model, we can plug in different set of transition rates to simulate different system environments. We report two sets of simulations: (a) one for a peer-to-peer file-sharing environment such as Maze [21], with $1/\lambda$ = 4.6 hours, $1/\mu$ = 12.3 hours, and $1/\delta$ = 58 days; (b) and one for a wide-area collaboration environment such as PlanetLab, with $1/\lambda$ = 8.5 days, $1/\mu$ = 3.5 days, and $1/\delta$ = 200 days. These parameters are obtained from actual system traces, in which we use 30-day as a threshold to determine a permanent failure.

For each setting, we use Monte-Carlo simulations to evaluate the bandwidth cost and availability of three classes of detection/recovery mechanisms: (a) timeout-based detection; (b) Protector; and (c) an oracle detector. Each run simulates 2000 data objects randomly scattered at start time across 1000 nodes.

The initial number of replicas for each object is the target replica threshold t_r, which is computed as follows. We first derive node availability p_c, which in our Markov model is

$$p_c = \frac{\mu}{\lambda + \mu}, \qquad (9)$$

since $1/\lambda$ is the mean time a node stays in state A before it transitions to T while $1/\mu$ is the mean time it stays in T before it transitions back to A. Then we use (9) and (1) to obtain the target replica threshold t_r.

We assign target availability p_o in each setting as follows: p_o = 0.923 for the Maze-like environment, and p_o= 0.9927 for the PlanetLab-like environment. We use a lower target availability level for the Maze-like system because the system is more dynamic and it is more time-consuming to simulate such a dynamic system maintaining a higher level of data availability. The particular values we choose for the target availability guarantees that when computing the target number of replicas t_r using equation (1), the value before taking the ceiling operation is already an integer (8 for the Maze-like environment and 4 for the PlanetLab-like environment). This eliminates the artificial boosting of the availability when taking the ceiling operation to obtain t_r.

We simulate replica maintenance for each class of detectors. We run each simulation for a long simulated time period (100 days for the P2P environment and 300 days for the PlanetLab-like environment) and collect the synthetic trace. The actual availability is obtained by sampling object availability at regular time intervals (one hour) to compute the ratio between the number of sampled time points at which the object is available and the total number of sampled points, then taking the average over 2000 objects. The bandwidth cost is computed by obtaining the total number of replicas recovered (excluding the initial number t_r of replicas) divided by the time period and by the number of objects. Since permanent failures reduce the number of nodes in the system, we also add a random process to add new nodes into the system with the same rate as permanent failures.

For timeout detectors, we simulate different timeout values. Whenever a timeout causes the number m of remaining replicas fall below the threshold t_r, we generate $t_r - m$ replicas randomly on remaining online machines. For Protector, we use equation (3) to compute $F_i(d)$, then simulate the behavior of Protector using equation (2) and the description in Section 2.2. For the oracle detector, we assume that it knows exactly whether a failure is transient and permanent, and only generates new replicas for permanent failures.

Figure 3 shows the first set of results for the peer-to-peer system setting. The x-axis represents unavailability (*i.e.* 1 – availability) to make the cost-availability tradeoff clear. The y-axis increases with cost, while the x-axis increases as availability decreases. Timeout detectors ranging from 10 hours to 120 hours are simulated. The results show that when timeout is set to be very small, both availability and recovery costs are very high. But when the timeout is set to be large, the availability falls below the target availability even though the recovery cost is small.

Protector provides a good balance between availability and recovery cost. Its availability is slightly better than the target level (actual availability is 0.9298, 1% higher than the target availability). Its cost is low and close to the best results achieved by either the oracle detector or the best possible timeout detector, *e.g.* Protector's cost is 0.1520 copy per object per day, 6.6% higher than the cost of the oracle detector. Given the same availability level, we see that Protector achieves lower cost than timeout detectors. There is a small window of timeout values in which timeout de-

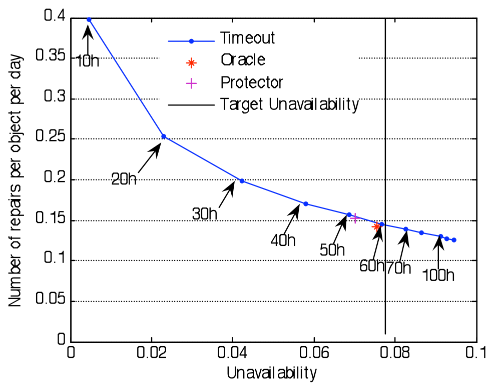

Figure 3. Recovery cost versus availability tradeoff with the synthetic trace for the Maze-like environment.

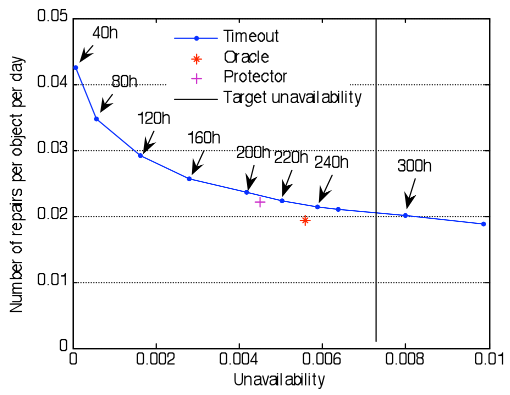

Figure 4. Recovery cost versus availability tradeoff with the synthetic trace for the PlanetLab-like environment.

tectors meet the availability target while incurring slightly lower cost than Protector. However, this window is small (from 55 hours to 60 hours), and the bandwidth saving is not significant (the best timeout detector only saves 4.1% over Protector in this window).

Comparing to the oracle detector, we see that Protector provides better availability at a slightly higher cost. This is because Protector still makes wrong estimates, and when it wrongly omits one recovery action (false negatives), it could compensate in the next estimate interval, but when it wrongly recovers some replicas (false positives), there is no compensation and the detector simply wait for the number to drop. Thus Protector leans slightly towards more replication and higher availability.

The oracle detector has the lowest recovery cost while still maintaining the target availability. The reason why it has slightly better availability than the target level is that whenever it detects a permanent failure, it recovers a new replica on an *online* node. This is better than the behavior when all failures are transient (in which t_r is calculated), because in the latter case a replica is in a transiently failed node when a failure occurs, but with the oracle detector a new replica is generated at an online node instead of at a transiently failed node.

Finally, we see that while Protector can use a conservative 30-day threshold to estimate *MTTF*, *MTTR*, and *MLT* and achieve the desired availability level, the timeout-based detector cannot use 30-day as the timeout: any timeout larger than 60 hours fails to achieve the availability goal. This again shows that timeout-based detectors require finely tuned timeouts, while Protector can start with very conservative timeouts and still achieve near-optimal availability and cost tradeoff.

Our results are taken from a large sample size (2000 objects), and confidence intervals for our results are very small and omitted from the figures (*e.g.* for the availability provided by Protector in Maze-like environment, its mean is 0.9298, and its 95% confidence interval is [0.9294, 0,9302]).

The results from PlanetLab in Figure 4 show similar trends as the ones in Figure 3. Protector provides availability slightly higher than the target availability and the availability provided by the oracle detector. Given the same availability level, Protector has a smaller recovery cost than the timeout detector. The bandwidth cost of Protector is only slightly higher than that of the oracle detector and that of the best-tuned timeout detector.

3.2. Evaluation using System Traces

In this section, we use actual system traces collected from a peer-to-peer file-sharing system to evaluate the performance of the Protector as compared with timeout-based detectors and the oracle detector. The peer-to-peer system that we use is Maze [21], which we now briefly explain.

With an average of 20K simultaneously online nodes, Maze [21] is one of the largest distributed systems on CERNET (China Education and Research Network). Maze can be a representative of highly dynamic distributed system since its dynamic level is close to Overnet [19]. We choose it as our experiment environment mainly because it is a large, distributed collection of hosts that has been monitored for long periods of time. We construct the environment with the system log from 3/1/2005 to 5/31/2005. To obtain the mean life time of a node, we use a method similar to [22] and estimate that MLT = 58 days. We use 30-day as the threshold to determine a permanent failure.

We use 3000 nodes in the system to store 2000 objects. The historical behavior of all nodes in March is used to predict the behavior of nodes in April. The

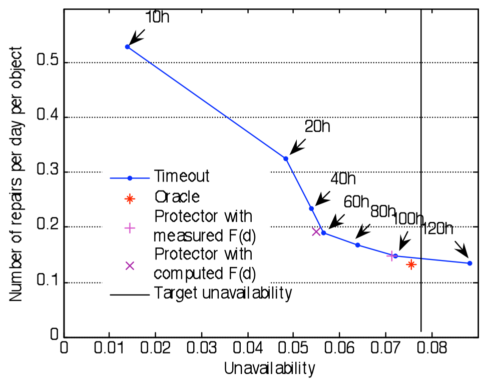

Figure 5. Recovery cost versus availability by the real trace-driven simulation in the Maze environment

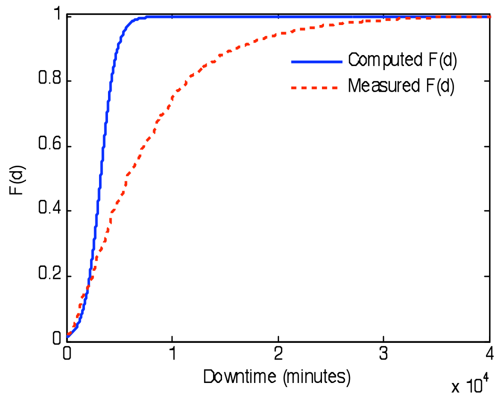

Figure 6. Comparison between computed *F*(*d*) and measured *F*(*d*).

probability function $F(d)$ is estimated using two methods we proposed in Section 2.4 and 2.5, respectively. In particular, we first use the Markov model (*i.e.* equation (3)) with the three parameters (*MTTF*, *MTTR*, *MLT*) the same as in Section 3.1 to compute $F(d)$. We call this $F(d)$ *computed* $F(d)$. Next we use the trace data in March with equation (8) to estimate $F(d)$, and we call this $F(d)$ *measured* $F(d)$. Given an object and its target number t_r of replicas, the system first places t_r replicas on a group of nodes available on April 4th. Then the simulator added and removed nodes based on their availability in the trace. Protector estimates the number m of replicas remaining in the system every hour. If $m<t_r$, Protector triggers data recovery and regenerates t_r-m replicas on other t_r-m available nodes at that time.

The result of our simulation is summarized in Figure 5. An important observation is that Protector using computed $F(d)$ values results in higher availability and higher cost, comparing with Protector using measured $F(d)$ values (1.7% higher availability with 30% higher cost). The main reason is that actual distributions of time to failures and time to recovery are not exponential. In particular, the results in [19] already show that the distribution of time to recovery (*TTR*) tends to have heavier tail than the exponential fit. This means that the actual transient failures may take longer time to recover. Thus, when using the actual *TTR* distribution to estimate $F(d)$ given a down time of d time units of a node, one would have a lower probability to declare that the node is permanently failed than the case when $F(d)$ is computed based on the exponential distribution assumptions. In fact, Figure 6 clearly shows that computed $F(d)$ is in general higher than measured $F(d)$, which means that Protector with computed $F(d)$ is more aggressive in declaring a failure as permanent and starting the recovery task. Therefore, Protector with computed $F(d)$ provides a higher availability at a higher recovery cost. However, quantitatively, even the Protector with computed $F(d)$ provides reasonably good tradeoff between availability and the recovery cost: its recovery cost is 0.1928 copy per object per day, which is 46% higher than the oracle detector (0.1324 copy per object per day) and is 30% higher than the close-to-optimally tuned timeout detector (0.1479 copy per object per day with timeout value of 100 hours). Therefore, for applications comfortable with such a cost increase, using Protector with the easily computed $F(d)$ is good enough. If one wants to further reduce the recovery cost, one can consider using the actual traces to measure $F(d)$. Our results show that this will give very close to optimal cost (0.1487 copy per object per day, 12% higher than the oracle, and 0.5% higher than the best-tuned timeout detector).

Our extensive simulations show that using a simple Markov model, Protector provides a good tradeoff between availability and recovery costs. Using failure data from past events, Protector can achieve near-optimal tradeoffs only achievable by the oracle detector. In contrast to timeout-based detectors that require optimal parameter tuning to achieve good results, Protector achieves its performance without any parameter tuning.

4. Discussion

Comparing against the extra replica method. In Section 2, we clearly showed that Protector's estimation method is the best among all methods that estimate the number of remaining replicas in the system, which include timeout-based methods. As an alternative, recent studies [5, 6, 13] decrease the cost due to transient failures by proactively adding several extra replicas. The system does not trigger data recovery until all extra replicas are lost. By this way, the system delays the creation of new replicas and masks some transient failures. We show below that the extra replica method is orthogonal to our Protector's estimation method, and Protector may help the extra replica method to overcome its own issues.

A system using the extra replica method still needs to monitor the nodes containing the replicas and obtain an estimate on the remaining number of replicas. It needs to use this estimate to decide if additional replicas are necessary. Its advantage is that it delays data recovery so that many transient failures can be masked without generating new replicas. However, existing studies still use timeouts to estimate the number of remaining replicas, which has been shown by our analysis in Section 2 be to suboptimal. Instead, we can use Protector's estimation method to give the best estimate on the remaining replicas in the system. Our proposition in Section 2 shows that this will be better than any timeout-based estimation method. Therefore, Protector can act in unison with extra replication to mask transient failures with minimal cost.

Avoiding periodic calculation in Protector. Our basic Protector algorithm requires that Protector periodically estimate the number of remaining replicas in the system. One may question that this is still a parameter that Protector may need to configure. The answers to such questioning are twofold. First, this parameter is not critical, and shortening the period only increases the cost of CPU computation, which is usually not the bottleneck resource in distributed storage systems. Second, the Protector may be further adapted to eliminate even this parameter, as explained below.

Instead of periodic calculations, Protector requires computation only after node failure and recovery events. Upon each of these events, Protector computes the time point in the future when the number of remaining replicas would drop below the target level based on the function $F(d)$, provided that no more failures and recoveries occur. When this time point is calculated, it invalidates the previously calculated time points based on earlier events and sets a timer to trigger a data recovery at this future time point. The calculation of this future time point can be computationally intensive, but it may save periodic calculations and can further be optimized using techniques such as binary search schemes on future time periods and/or pre-computed reference tables. With this modification, Protector will be truly parameter-free to applications.

5. Related Work

There is a significant work on the study of failure detectors in networked systems. In particular, many studies investigate the quality of service of failure detectors and adaptive failure detectors [15-17, 20, 23-25]. These studies also use probabilistic approach in the design and analysis of the failure detectors. However, there are important differences between these failure detectors and Protector. First, the objectives of the failure detectors are different. While these failure detectors focus on providing fast and accurate detection of all failure events (transient and permanent), Protector is designed specifically to distinguish permanent failures from transient failures.

Second, while these failure detectors mainly rely on message delay distributions to adjust the tradeoff between detection speed and accuracy, Protector relies on the distribution of permanent failures vs. transient failures to adjust the tradeoff between high availability and low maintenance cost. Third, while these failure detectors focus on detecting individual node failures, Protector works on the replica-group level to achieve higher reliability with lower cost. Overall, these approaches are complementary and can be used together: we can use a QoS-based failure detector to detect all failure events, and then use Protector to decide when to start data recovery based on the downtime of nodes in a replica group.

Many wide-area storage systems use timeouts to detect permanent node failures, but it is well known that the task is difficult. Several projects [5, 6, 13] mask transient failures and delay triggering data recovery by proactively adding a number of extra replicas. Weatherspoon et al. studied the increased costs due to transient failures [13]. Their results quantify the benefits of maintaining extra replicas to reduce these transient costs. However, this approach still performs extra replication that may not be necessary to maintain the desired level of availability. In [9], Chun et al. suggests that reintegrating returning replicas is the key to avoiding unnecessary replication. An alternative to using extra replicas is to increase the timeout threshold as suggested by [26]. All these approaches are timeout based and are on individual node detections. Our protector, on the contrary, avoid completely the tuning of timeouts, and is based on the entire replica group, which gives us better performance in achieving high availability with a low maintenance cost.

Other work tries to detect permanent failures by assuming that system behavior *prior* to a permanent failure may deviate from its normal behavior, such as increased failure frequency [27] or statistical changes report by the S.M.A.R.T. technology for disks [28]. Protector is different in that it relies only on the behavior of nodes after they fail, so it works even for systems such as peer-to-peer storage systems where the above assumption does not hold. When the assumption does hold, Protector can also consider the abnormal behaviors prior to failures to increase the detection accuracy.

6. Conclusion

Detecting permanent failures is the key to guarantee high data availability while minimizing system maintenance bandwidth. We have presented a novel methodology that estimates the number of remaining replicas rather than determining if a single failure is permanent or transient. We have shown that Protector provides the best estimate on the number of remaining replicas in the system among all methods including the timeout-based methods. We believe Protector will complement the extra replica method to mask transient failures while reducing the cost of maintaining extra replicas. Both model-based and trace-driven simulations show that Protector achieves performance close to that of the perfect oracle detector and outperforms most timeout-based detectors. We demonstrate how to tune Protector based on history without setting manual parameters.

There are several directions to extend the current study. First, we can study this problem as an optimization problem based on the Markov model proposed in the paper. Given the availability target, under our Markov model, what recovery strategy should we use to minimize the recovery cost? We have tried the probabilistic method of the Protector, and compared it with the traditional-timeout based method, but there could be other methods that yield the optimal cost and availability tradeoff. Another direction is on how to accommodate the heterogeneous behaviors of nodes in the system. The basic Protector method proposed in this paper considers that all nodes in the system have homogeneous failure and recovery behaviors. Actual systems may exhibit more heterogeneous behaviors.

References

[1]S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *Proc. of SOSP,* pp. 29-43, 2003.

[2]"Amazon Web Services. http://s3.amazonaws.com."

[3]P. Lenssen, "Google Gdrive Surfaces. http://blog.outer-court.com/archive/2006-07-10-n48.html.," 2006.

[4]J. Kubiatowicz, C. Wells, B. Y. Zhao, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, and S. Rhea, "OceanStore: an architecture for global-scale persistent storage," *Proc. of ASPLOS,* 2000.

[5]K. Tati and G. Voelker, "On object maintenance in peer-to-peer systems," *Proc. of IPTPS,* 2006.

[6]R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker, "Total Recall: System Support for Automated Availability Management," *Proc. of NSDI,* 2004.

[7]A. Adya, R. Wattenhofer, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, and M. Theimer, "Farsite: federated, available, and reliable storage for an incompletely trusted environment," *Operating Systems Review,* 2002.

[8]Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin, "BitVault: a Highly Reliable Distributed Data Retention Platform," *Operating Systems Review,* 2007.

[9]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," *Proc. of NSDI,* 2006.

[10]F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," *Proc. of SOSP,* 2001.

[11]P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," *Proc. of HotOS,* 2001.

[12]C. Blake and R. Rodrigues, "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two," *Proc. of HotOS,* 2003.

[13]H. Weatherspoon, B.-G. Chun, C. So, and J. Kubiatowicz, "Long-Term Data Maintenance in Wide-Area Storage Systems: A Quantitative Approach," *IEEE Computer,* 2005.

[14]J. Tian, Z. Yang, and Y. Dai, "A Data Placement Scheme with Time-Related Model for P2P Storages," *Proc. of IEEE P2P,* 2007.

[15]L. Falai and A. Bondavalli, "Experimental evaluation of the QoS of failure detectors on wide area network," *Proc. of DSN,* 2005.

[16]W. Chen, S. Toueg, and M. K. Aguilera, "On the Quality of Service of Failure Detectors," *IEEE Trans. on Computers,* pp. 561-580, 2002.

[17]M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," *Proc. of DSN,* 2002.

[18]I. Sotoma and E. R. M. Madeira, "Adaptation-algorithms to adaptive fault monitoring and their implementation on CORBA," *Proc. of DOA,* 2001.

[19]J. Tian and Y. Dai, "Understanding the Dynamic of Peer-to-Peer Systems," *Proc. of IPTPS,* 2007.

[20]N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The Φ Accrual Failure Detector," *Proc. of SRDS,* 2004.

[21]M. Yang, B. Y. Zhao, Y. Dai, and Z. Zhang, "Deployment of a large scale peer-to-peer social network," *Proc. of WORLDS,* 2004.

[22]L. Ni and A. Harwood, "A comparative study on Peer-to-Peer failure rate estimation," *Proc. of ICPADS,* 2007.

[23]A. Casimiro, P. Lollini, M. Dixit, A. Bondavalli, and P. Veríssimo, "A framework for dependable QoS adaptation in probabilistic environments," *Proc. of ACM SAC*, 2008.

[24]R. C. Nunes and I. Jansch-Porto, "QoS of Timeout-Based Self-Tuned Failure Detectors: The Effects of the Communication Delay Predictor and the Safety Margin," *Proc. of DSN,* 2004.

[25]M. Bertier, O. Marin, and P. Sens, "Performance analysis of a hierarchical failure detector," *Proc. of DSN,* 2003.

[26]R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication," *Proc. of IPTPS,* 2005.

[27]A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni, "Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults," *IEEE Trans. on Computers,* pp. 230-245, 2000.

[28]Seagate, "Playing it S.M.A.R.T. --- Self Monitoring. Analysis and Reporting Technology (S.M.A.R.T) Freq uently Asked Questions", http://www.seagate.com/supp ort/kb/disc/smart.html.