Protector: A Probabilistic Failure Detector for Cost-Effective Peer-to-Peer Storage

Zhi Yang, Jing Tian, Ben Y. Zhao, Wei Chen, and Yafei Dai, *Member*, *IEEE*

Abstract—Maintaining a given level of data redundancy is a fundamental requirement of peer-to-peer (P2P) storage systems—to ensure desired data availability, additional replicas must be created when peers fail. Since the majority of failures in P2P networks are transient (i.e., peers return with data intact), an intelligent system can reduce significant replication costs by not replicating data following transient failures. Reliably distinguishing permanent and transient failures, however, is a challenging task, because peers are unresponsive to probes in both cases. In this paper, we propose Protector, an algorithm that enables efficient replication policies by estimating the number of "remaining replicas" for each object, including those temporarily unavailable due to transient failures. Protector dramatically improves detection accuracy by exploiting two opportunities. First, it leverages failure patterns to predict the likelihood that a peer (and the data it hosts) has permanently failed given its current downtime. Second, it detects replication level across groups of replicas (or fragments), thereby balancing false positives for some peers against false negatives for others. Extensive simulations based on both synthetic and real traces show that Protector closely approximates the performance of a perfect "oracle" failure detector, and significantly outperforms time-out-based detectors using a wide range of parameters. Finally, we design, implement and deploy an efficient P2P storage system called AmazingStore by combining Protector with structured P2P overlays. Our experience proves that Protector enables efficient long-term data maintenance in P2P storage systems.

Index Terms—Failure detector, P2P storage, availability, replication management.

1 Introduction

Peer-to-peer (P2P) storage networks aim to aggregate today's resource-abundant computers to form large, decentralized storage systems. Of the numerous P2P storage systems developed in recent years, prototypes including CFS [14], TotalRecall [7], Friendstore [33] and commercial services like Wuala [3] and CleverSafe [2], all of them faced the long-standing problem of enforcing required data availability from participating peers.

To reach availability target in unreliable networks, storage systems replicate data across multiple peers (using replication or erasure coding). Even if some replicas (or fragments) become unavailable due to their hosts failing, the object is still available by accessing other online replicas.[1] Over longer periods, the system must generate new replicas as needed to compensate for others lost to peer failures. Since replica generation consumes a large amount of bandwidth, maintaining availability incurs a heavy cost on storage systems [8]. Intuitively, aggressive replication incurs high bandwidth costs that may cripple the entire system, while reducing the number of replicas generated might result in an unacceptably low level of availability. Therefore, the challenge in building P2P storage systems is to carefully navigate this cost-availability trade-off, by reducing maintenance cost as much as possible while maintaining the desired level of availability.

This challenge is further complicated by the fact that peer failures can be either transient or permanent. Data are lost following a permanent failure, and the expected level of redundancy must be restored by creating new replicas. In contrast, an object with sufficient replicas can tolerate transient failures without sacrificing availability, since a peer undergoing a transient failure will rejoin the network and bring back its stored data. Hence, an ideal system would identify peer failures and only replicate data following permanent ones. Unfortunately, reliably distinguishing permanent and transient failures turns out to be a daunting task, since they cannot be distinguished using probes [12], [34].

To address this problem, this paper proposes Protector, an algorithm that provides better protection against transient failures. In particular, given a replica group (i.e., all peers hosting replicas of the same object) and downtime of these peers, Protector can detect the number of *remaining replicas* in the group, i.e., the number of replicas residing on online peers or peers experiencing transient failures. Storage systems can use this detected number of replicas to determine whether data recovery is necessary to reach the desired replication level.

In essence, Protector drastically improves detection accuracy by exploiting two opportunities: First, we obtain a failure probability function $F(d)$ that represents the

1. For simplicity, we will use the terms *replicas* and *fragments* interchangeably in the remainder of this paper.

- *Z. Yang, J. Tian, and Y. Dai are with the Department of Electrical and Computer Engineering, Peking University, Room 1717E, Science Building 1, No. 5 Yiheyuan Road, Haidian District, Beijing, P.R. China 100871. E-mail: {yangzhi, tianjing, dyf}@net.pku.edu.cn.*
- *B.Y. Zhao is with the Department of Computer Science, University of California, Santa Barbara, Santa Barbara, CA 93106-5110. E-mail: ravenben@cs.ucsb.edu.*
- *W. Chen is with Microsoft Research Asia, Sigma Building 5/F, 49 Zhi Chun Road, Haidian District, Beijing, China 100190. E-mail: weic@microsoft.com.*

Manuscript received 21 Mar. 2010; revised 14 July 2010; accepted 16 Sept. 2010; published online 9 Nov. 2010.
Recommended for acceptance by Y. Hu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-03-0166.
Digital Object Identifier no. 10.1109/TPDS.2010.205.

1045-9219/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

probability that a peer has permanently failed given an observed failure of d time units. This exploits the downtime difference among failures, thus providing a more accurate indication of the failure type. Further, we aggregate failure probabilities across all peers in the replica group to derive an estimate of the number of remaining replicas, balancing false positives for some peers against false negatives for others.

We make four major contributions in this paper.

- First, we propose Protector, a probabilistic group-based algorithm for guiding efficient replication. After that, we prove that among all techniques based on peer downtime (including time-out-based methods), standard Protector achieves optimal accuracy in estimating the number of remaining replicas.
- Second, in order to avoid high computation cost in cases where the replica group size is large, we also present an approximate Protector which can achieve a good tradeoff between accuracy and complexity.
- Third, we compare the effectiveness of Protector against time-out-based techniques through simulations using both synthetic and real traces. Our results show that not only does Protector outperform alternative approaches for almost all time-out parameters, it also approximates the performance of the ideal oracle failure detector.
- Fourth, using Protector algorithm together with structured P2P overlay, we design, implement, and deploy an efficient P2P online storage system called AmazingStore [1]. We report Protector's performance from real-world measurement.

This paper extends an earlier version [31] in several substantial ways. First, we use a semi-Markov process (SMP) to predict peer failure probability, leading to a more widely applicable solution. Second, we recognize that Protector incurs $O(2^n)$ worst-case time complexity, and propose an "approximate Protector" that scales to large replica group sizes (n) without significantly degrading accuracy. Third, we present more performance details of Protector, including its performance under coding schemes. Fourth, we report on our long-term experiences with an implementation of Protector in AmazingStore [1], a real P2P storage system that has been running for eight months with a daily peak of 1,000 simultaneous online peers. Finally, we demonstrate that Protector is also useful in systems that adopt a proactive repair strategy.

The rest of this paper is organized as follows: First, we describe related work in Section 2. Next, Section 3 describes the Protector methodology and its features in detail. Section 4 evaluates Protector through both synthetic and real trace-driven simulations. Section 5 presents the deployed P2P storage prototype and reports Protector's real-world performance. Section 6 discusses the usefulness of Protector under proactive replication model, and finally we conclude the paper in Section 7.

2 Related Work

2.1 Protector versus QoS-Based Failure Detector

There is a large amount of work on failure detection in networked systems. In particular, many studies investigate the quality of service of failure detectors and adaptive failure detectors [4], [5], [10], [11], [16], [21]. 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 the 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), our 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 trade-off between detection speed and accuracy, Protector relies on the distribution of permanent failures versus transient failures to adjust the trade-off 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 availability 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 peers in a replica group.

2.2 Protector versus Extra Replica Method

While not explicitly addressed by early systems, recent research work generally adopts one of two approaches: lazy replication [7], [29], [34] and time-out thresholds [8], [12], [24]. In lazy replication, the system masks transient failures and delays triggering data recovery by proactively adding a number of extra replicas. Weatherspoon et al. studied the increased costs due to transient failures [34]. Their results quantify the benefits of maintaining extra replicas to reduce transient costs, but determining the number of extra replicas is nontrivial. In [12], Chun et al. suggested that reintegrating returning replicas can dynamically add suitable number of extra replicas. However, these methods still incur significant extra replication cost in P2P environments where peers are prone to failures. Our Protector method is orthogonal to the extra replica method, moreover, Protector can help the extra replica method to further differentiate failures among unavailable peers and thus reduce the number of extra replicas needed to mask transient failures.

2.3 Protector versus Time-out Threshold Approach

An alternative to adding many extra replicas is to increase the time-out threshold as suggested by [24]. In the time-out threshold approach, peers wait for some preset time-out period before identifying a failure as permanent and acting to restore replication levels. While simple, this approach is error-prone: 1) it takes downtime differences into consideration but at a coarse-grained level; and 2) choosing the time-out parameter walks a fine line between high values that produce false negatives and low values that produce false positives. Our Protector, on the contrary, overcomes these issues using a probabilistic and group-based method, which achieves high availability with a low maintenance cost.

2.4 Protector versus Abnormality Detection

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 [9] or statistical changes report by the S.M.A.R.T. technology for disks [26]. Protector is different in that it relies only on the behavior of nodes after they fail, so it works even for systems such as P2P storage systems where the above assumption does not hold. When the assumption does hold, Protector can certainly consider the abnormal behaviors prior to failures to increase the detection accuracy.

2.5 Protector versus Proactive Replication

Instead of reactive replication method (e.g., replacing lost redundancy in reaction to failures), some systems adopt proactive replication method (e.g., constantly creating additional redundancy with a narrow background bandwidth) to mask transient failures (e.g., Tempo [27] and Glacier [18]). However, this approach is still error-prone since it has to carefully tune the replication bandwidth. Moreover, proactive replication brings a new design issue on deciding replication priority, since it may take a long time to produce a new replica with small repair bandwidth. As will be discussed in Section 6, our Protector's estimation method can help proactive replication to give priority to objects with fewer remaining replicas, since they are more likely to be destroyed by unpredictable future failures. Besides, some proactive systems (e.g., [22]) try to reduce request latency by actively replicating popular objects, but they ignore the rejoin of peers. For these systems, Protector can be incorporated to maintain the optimal replication level efficiently.

3 Protector: Probabilistic Failure Detection

In this section, we shall describe the Protector approach in the specific context of the reactive data maintenance. We begin by introducing the high-level maintenance process after adopting Protector, and then describe in detail how to derive Protector's failure probability function through a semi-Markov failure model. After that, we describe two Protector approaches at different accuracy granularity (with performance/overhead trade-off). We finally extend Protector to a class of detectors, enabling a more general application scenario.

Below, we assume that peers are cooperative and their failures are independent, so that we can focus our attention on permanent failure detection. In systems where these assumptions are relaxed, tamperproof mechanisms [13] and selective placement schemes [35] can be further employed to make Protector less sensitive to these assumptions.

3.1 Protector-Based Reactive Maintenance

Protector is designed to guide the replication decisions of an available P2P storage system. In this paper, we mainly focus on systems operating under the *reactive* replication model, because most existing P2P storage systems belong to such a case. These systems seek to maintain a target replication level t_r for objects at all times. Once the system detects the replication level falls below t_r, it reacts by creating new replicas to bring the replication level back to t_r. More specifically, our maintenance scheme works as follows:

The replication target t_r can be determined as a function of the desired level of data availability (as in [7], [12], [32], [34]). Suppose the target availability is $p_o \in (0,1)$, and the availability of each individual host peer (before the host fails permanently) is p_c. When using a replication scheme, 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 $1-(1-p_c)^x$. 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:

$$t_r = \frac{\log(1-p_o)}{\log(1-p_c)}. \tag{1}$$

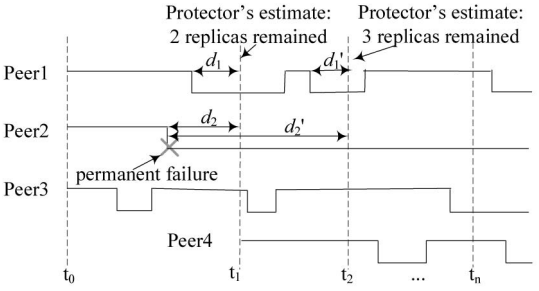

Fig. 1. Reactive data recovery with Protector ($t_r = 3$).

An alternative technique that has been proposed by several systems is the use of erasure coding [7], [23], [24]. Under this scheme, each object is divided into b blocks which are then stored with an effective redundancy factor k_c. The object can be reconstructed from any available b fragments taken from the stored set of $k_c b$ fragments. Object availability is given by the probability that at least b out of $k_c b$ fragments are available, that is $\sum_{i=b}^{k_c b} \binom{k_c b}{i} p_c^i (1-p_c)^{k_c b - i}$. Using algebraic simplifications and the normal approximation to the binomial distribution (see [24]), we get the target number t_r of fragments (which is the minimum $k_c b$ making data availability meet the target):

$$t_r = \left(\frac{\sigma \sqrt{\frac{p_c(1-p_c)}{b}} + \sqrt{\frac{\sigma^2 p_c (1-p_c)}{b} + 4p_c}}{2p_c} \right)^2 b. \tag{2}$$

For each object, given a global target replication level t_r, a master responsible for this object periodically (e.g., every 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 peers; if $m \geq t_r$, no actions are necessary. Fig. 1 shows a simple example with $t_r = 3$. At time t_1, only peer 3 is up and Protector estimates that there are two replicas remaining in the system, and thus the system generates a new replica on peer 4. Later at time t_2, peers 1 and 2 are down, peers 3 and 4 are up, and Protector estimates that there are three remaining replicas (i.e., a replica on either peer 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

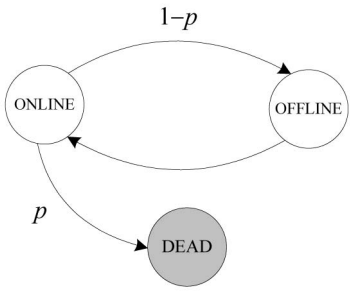

Fig. 2. The embedded Markov chain model of a peer life.

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 section, we focus on the detection and data recovery algorithm. We will present the actual implementations in Section 5.

3.2 Failure Probability Function

The effectiveness of Protector revolves around the function $F(d)$, the conditional probability that a peer permanently fails given that it has already been unavailable for d time units.

More formally, let TTR_i be the random variable representing "time-to-recover" of peer i, that is, the time from when peer i fails to the time when it recovers. TTR_i could take value ∞, which means peer i has failed permanently, and will never recover. Let d_i be the downtime of a peer i, and $d_i = 0$ if peer i is online (e.g., d_1 and d_2 in Fig. 1 for peers 1 and 2 for the estimate at time t_1). The downtime of a peer begins when a low-level failure detector declares a failure of a peer based on message delay or lack of responsiveness. Such failure detectors have been extensively studied (e.g., [4], [11], [16]). When $d_i > 0$, we define function $F(d_i) = P(TTR_i = \infty | TTR_i > d_i)$. Finally, $F(0) = 0$, i.e., if peer i is online, it cannot be a permanent failure.

To derive function $F(d_i)$, we characterize the failure and recovery behaviors of a peer using a continuous semi-Markov chain [25]. Each peer always begins its existence in an online state, then alternates between online and offline states during its lifetime, and finally enters a dead state. Unlike standard Markov chains used in [12], [15], the time spent in online and offline states follows a general distribution. Thus, a semi-Markov process is not Markovian at an arbitrary point of time. However, one can create an embedded Markov chain by sampling the original process at the moments of transition to a new state.

Fig. 2 depicts the embedded Markov chain of peer behavior. When a peer disconnects, it can go either temporarily or permanently offline. The actual event is governed by the probability p, which is the conditional probability that a peer goes to dead state given that it is leaving the online state, that is, $p = P(TTR_i = \infty | TTR_i > 0)$. Since $TTR_i > 0$ is always true, we can rewrite p as $p = P(TTR_i = \infty)$.

This probability can also be expressed in terms of well-known parameters that describe the dynamic behavior of peers, including 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 peer. From the model, we see that peer lifetime is the absorption time of the chain in dead state. If the peer generates N number of transient failures before it fails permanently (e.g., the chain visits offline states N times before it reaches dead state), we have

$$MLT = E[N](MTTF + MTTR) + MTTF. \tag{3}$$

Notice that N follows a geometric distribution, hence, $E[N] = (1-p)/p$. Since a peer's MLT is usually sufficiently larger than its MTTR in practical systems, we can solve p from (3) and obtain

$$p = \frac{MTTF + MTTR}{MLT}. \tag{4}$$

With the semi-Markov model, a simple application of the Bayesian's Theorem gives the function $F(d_i)$:

$$F(d_i) = \frac{p}{p + (1-p)P(TTR > d_i)}. \tag{5}$$

Here, $P(TTR > d_i)$ is the complementary cumulative distribution function (CCDF) of TTR.

When actual failure traces are available, we can derive parameters p and $P(TTR > d_i)$ directly from the measurement trace. In particular, we monitor the failure (disconnection) and recovery (reconnection) behaviors of all peers for a period of time (e.g., one month), and obtain

$$p = 1 - \frac{\#of\ Reconnections}{\#of\ Disconnections}, \tag{6}$$

and $P(TTR > d_i)$ is the fraction of reconnections whose spans are larger than d_i.

An important remark is now in order. In order to extract the quantities used for our calculation (e.g., # *of Reconnections*), we need to identify permanent failures in the historical traces. This essentially requires a threshold value T such that failures lasting longer than this threshold are classified as permanent failures. Although a larger threshold allows more peers to recover, it requires the measurement of longer period of trace and thus larger bootstrapping/adjusting delays. To make this trade-off, the system can look at the CCDF of TTR and derive a threshold where the curve is getting flat. This would make Protector less sensitive to the threshold since there will hardly exist reconnections after the threshold. For our experiments and prototype, we derive a 30-day threshold by setting $P(TTR > T) \leq 0.001$. We emphasize that this threshold cannot be used by a time-out-based failure detector to determine permanent failures online. Online detections cannot afford such a large time-out, since it may put data availability in danger. Our evaluation results show this distinction clearly.

3.3 Standard Estimation Function in Protector

Protector uses a probabilistic availability estimation algorithm that improves its prediction accuracy by aggregating its prediction across the entire replica group, i.e., all peers hosting replicas for a given object.

Suppose that at the time of the estimate, there are n total peers hosting replicas of the object, a subset of which may

be online. Let the n peers be numbered as 1, 2, ... n, and d_i be the downtime of peer i ($d_i = 0$ if peer i is online). As we discussed in Section 3.2, we assume that $F(d_i)$ is known for any d_i. Let X be a random variable representing the number of remaining replicas in the system at the time of estimation. Given $F(d_i)$ for all peers $i = 1, 2, \ldots, n$, we can compute the probability of $X = k$ as

$$P(X=k) = \sum_{\substack{S\subseteq\{1,\ldots,n\}\\ |S|=k}} \prod_{i\in S}(1-F_i(d)) \prod_{i\notin S} F_i(d), \tag{7}$$

which is the probability of finding exactly k out of n peers are online or experiencing transient failures.

Protector picks the most likely number of remanning replicas m, e.g., $m = \arg_{0\le k\le n}\max P(X=k)$, as the estimated number of remaining replicas in the system.

Optimality in accuracy. We now show that Protector's estimate is optimal in terms of *accuracy rate*, which is the probability that the estimate correctly matches the reality, compared to all estimation functions based on function $F(.)$ and current downtime d_i.

Ultimately, any estimation approach 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 time-out periods use a time-out threshold TO_i and mark a peer i as permanently failed when $d_i > TO_i$. An accurate function $F(.)$ may help the detector to derive a good time-out value, but in the end it uses time-outs to decide on the number of remaining replicas. Other methods may determine 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

$$\begin{aligned} r &= \sum_{k=1}^{n} P(X=k)p_k \\ &\le \sum_{k=1}^{n} \max_{i\in\{1,\ldots,n\}}\{P(X=i)\}p_k = P(X=m). \end{aligned} \tag{8}$$

Therefore, the accuracy rate of any estimate method cannot exceed $P(X = m)$, which is what Protector achieves. Hence, we have the following proposition.

Proposition. *Among all detectors with the knowledge of downtime d_i and function $F(.)$ for all peers 1 to n, standard 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 time-out scheme to find the best time-out threshold, ultimately it is still used to obtain the number of remaining replicas, i.e., excluding the number of replicas on peers to be judged as permanently failed by the time-out thresholds. Therefore, any time-out scheme is covered by the above proposition, and Protector performs as well or better than any time-out-based approach in terms of accuracy rate.

3.4 An Approximate Estimation Function

Given a replica group containing n peers, our standard Protector algorithm incurs a time complexity of $O(2^n)$ in the worst case, since it examines every possible combinations across the group. This high computational cost prevents its practical application to cases where group size (n) is large. We recognize that large n values can easily arise due to many factors such as high target replication level t_r, coding-based replication as well as returning replicas reintegration, especially in highly dynamic P2P systems. In cases where large replica groups are desirable, we propose an approximate estimation method.

We obtain an approximate estimation function of m which runs in time complexity that scales linearly as a function of group size n. In particular, we pick unavailable peers from the replica group and take the mean of their failure probabilities. Let n_u be the number of unavailable peers and $\overline{F}$ is their average failure probability, we can use the following approximation to compute the probability of $X = k$ (with accuracy/complexity trade-off):

$$P(X=k) = \begin{cases} 0, & k < n-n_u, \\ \binom{n_u}{k-n+n_u}\overline{F}^{n-k}(1-\overline{F})^{k-n+n_u}, & n-n_u \le k \le n. \end{cases} \tag{9}$$

This formula is easy to understand: since any currently available peer cannot be permanently failed, there are at least $n - n_u$ number of remaining replicas. Therefore, in cases of $k \ge n - n_u$, the probability that exactly k replicas remain in the system is the probability that exactly $k - (n - n_u)$ peers out of n_u unavailable peers have remaining replicas. Note that we use a binomial distribution $B(n_u, 1-\overline{F})$ to approximate this probability, so it usually reaches its largest value at $\lfloor (n_u+1)(1-\overline{F})\rfloor$, which is the *mode* of the binomial distribution. Thus, we can compute m that maximizes $P(X = m)$ using the following equation:

$$m = \begin{cases} n, & n_u = 0, \\ n - n_u + \lfloor (n_u+1)(1-\overline{F})\rfloor, & n_u > 0. \end{cases} \tag{10}$$

Like standard Protector, the approximate estimator returns m as the estimated number of remaining replicas. We demonstrate via simulations in the next section that approximate Protector eliminates the high computation cost of standard Protector while achieving similar accuracy levels.

In practical systems, one can adopt a **hybrid** scheme which uses approximate Protector by default, but switches to the standard one any time the group size n hits a lower threshold. We use a hybrid Protector with a threshold value of 7 in the prototype implementation we describe in Section 5.

3.5 Extensions of Protector

Protector's methodology uses a maximum a posteriori (MAP) estimation rule to derive an estimate, since its estimate is the most likely one that occurs in reality. However, other estimation rules could also be explored as alternatives, depending on how one defines the impact of estimation errors in different application scenarios. Here, we discuss possible Protector extensions for these scenarios.

Clearly, the performance of an estimator depends mainly on the amplitude distribution of the estimation errors. Let $\varphi(x)$ be the penalty function, the estimator picks the number $m (0 \le m \le n)$ given by

$$m = \arg\min_{0 \le k \le n} \sum_{i=1}^{n} \varphi(k-i)P(X=i), \quad (11)$$

and returns m as the estimated number of remaining replicas. A penalty function $\varphi(x)$ allows us to customize how estimation errors are penalized. In most cases, $\varphi(x)$ is nonnegative, nondecreasing, and satisfies $\varphi(0) = 0$. This penalty function can be tuned based on the needs of specific scenarios.

We give some examples of estimators based on different penalty functions. First, our basic Protector can be viewed as putting the same penalty weight on all nonzero errors (e.g., $\varphi(x)$ is a positive constant when $x > 0$). Second, if we take $\varphi(x) = |x|$, the penalty function puts relatively stronger weight on small errors and (11) reduces to the minimum mean absolute error (MMAE) estimation. By the median theorem for the Weber problem, the solution for m is the *weighted median* of the sequence $\{1, 2, \ldots, n\}$ with weight $\{P(X=1), P(X=2), \ldots, P(X=n)\}$. Finally, if we take $\varphi(x) = x^2$, the penalty function puts relatively stronger weights on larger errors, and (11) reduces to the minimum mean square error (MMSE) estimation. This is a convex optimization problem and the solution is the *weighted mean* of the sequence $\{1, 2, \ldots, n\}$ with weights $\{P(X=1), P(X=2), \ldots, P(X=n)\}$.

4 Simulation-Based Evaluation

We evaluate the performance of Protector through simulations based on both failure model and actual system measurement trace. The two sets of simulations serve to validate different aspects of the advantages of Protector. The simulations based on the failure model validate whether the optimality in estimating the number of remaining replicas in the system leads to well-balanced trade-off 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 failure model. Our results show that in both cases the cost-availability trade-off provided by Protector is very close to that of an oracle detector that always knows exactly if a failure is permanent or not.

4.1 Evaluation Using Failure Model

The advantage of model-driven simulations is that they allow 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. For simplicity, we use a Markov failure model like other studies (e.g., [15]), which assumes exponentially distributed session and downtime (note that the SMP model described in Fig. 2 can be applicable to general failure scenarios).

System environments. We report two sets of simulations: 1) one for a peer-to-peer file-sharing environment such as Maze [36], with $\mathrm{MTTF} = 4.6$ hours, $\mathrm{MTTR} = 12.3$ hours, and $\mathrm{MLT} = 58$ days; 2) and one for a wide area collaboration environment such as PlanetLab, with $\mathrm{MTTF} = 8.5$ days, $\mathrm{MTTR} = 3.5$ days, and $\mathrm{MLT} = 200$ days. These parameters are obtained either from the actual system trace, or the existing measurement [34].

Workload. For each setting, we use Monte-Carlo simulations to evaluate the bandwidth cost and availability of three classes of detection/recovery mechanisms: 1) time-out-based detection; 2) Our Protector method; and 3) an oracle detector. Each run simulates 2,000 data objects randomly scattered at start time across 1,000 peers.

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

$$p_c = \frac{MTTF}{MTTF + MTTR}. \quad (12)$$

Then, we substitute p_c and target availability p_o into (1) (under replication scheme) or (2) (under coding scheme with $b = 6$) to obtain the target replica threshold t_r. Under both schemes, we aim to obtain about one nine availability for the Maze-like environment and two nine availability 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 desirable to mask more transient failures, trading imperfect availability for low replication cost and high scalability [17].

Further, in order to eliminate the artificial boosting or depressing of the availability when taking the rounding operation to obtain an integer t_r, we revise the target availability p_o in each setting as follows: 1) $p_o = 0.895$ for the Maze-like environment and $p_o = 0.9927$ for the PlanetLab-like environment under replication scheme; 2) $p_o = 0.909$ for the Maze-like environment and $p_o = 0.9940$ for the PlanetLab-like environment under coding scheme. The particular values we choose for the target availability guarantees that when computing the target number of replicas t_r, the value before taking the rounding operation is already an integer (7 for the Maze-like environment and 4 for the PlanetLab-like environment under replication scheme, 32 for the Maze-like environment and 14 for the PlanetLab-like environment under coding scheme).

Simulators. We simulate replica maintenance for each class of detectors. For time-out detectors, we simulate different time-out values. Whenever a time-out causes the number m of remaining replicas to fall below the threshold t_r, we generate $t_r - m$ replicas randomly on the remaining online machines. For Protector, we use (4) and (5) to compute $F(d_i)$, then simulate the behavior of Protector as described in Section 3. 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. Since permanent failures reduce the number of peers in the system, we also add a random process to add new peers into the system with the same rate as permanent failures.

We run each simulation for a long simulated time period (three months) 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 2,000 objects. The bandwidth

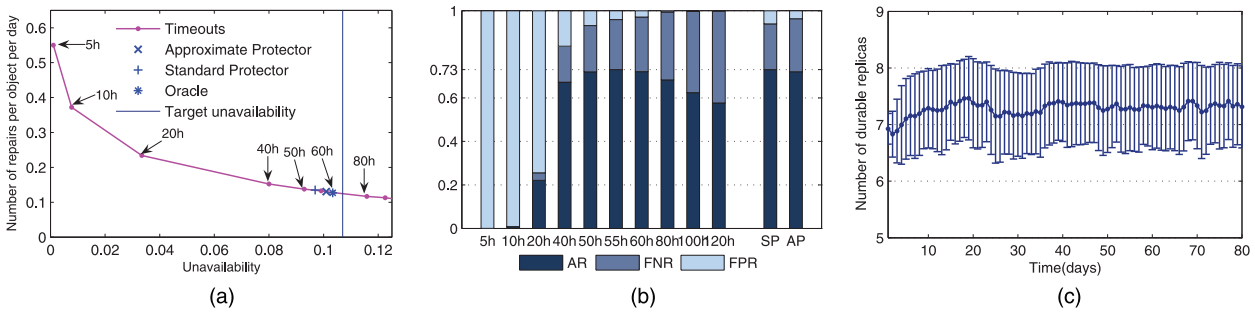

Fig. 3. Performance of standard Protector (SP) and approximate Protector (AP) with the synthetic trace for the Maze-like environment (with replication). (a) Cost-availability trade-off. (b) Percentage of availability estimates that are accurate, false positives, and false negatives. (c) Replica maintenance with standard Protector ($t_r = 7$).

cost is computed by obtaining the total number of replicas recovered divided by the time period and by the number of objects (In our simulation, we assume that a system using coding keeps an additional complete copy for each object, e.g., Wuala [3]).

Results. Figs. 3 and 4 show the evaluation results for the Maze-like system under replication and coding schemes, respectively. We summarize these results as follows:

Result 1: While time-out-based detector exhibit error-prone behavior, Protector automatically balances cost and availability. Figs. 3a and 4a show the cost-availability trade-offs different detectors achieve under each scheme. The x-axis represents unavailability (i.e., 1-availability) to make the cost-availability trade-off clear. The y-axis increases with cost, while the x-axis increases as availability decreases. Time-out detectors ranging from 5 to 120 hours are simulated.

The results show that time-out-based detectors are error-prone: when time-out is set to be very small, both availability and recovery costs are very high. But when the time-out is set to be large, the availability falls below the target availability even though the recovery cost is small. Few existing systems are able to select the right time-out value. Moreover, the time-out should be adjusted accordingly if different replica targets or replication methods are used, e.g., comparing Fig. 3a with Fig. 4a shows that the best-tuned time-outs are quite different under two schemes (60 and 50 hours, respectively).

In contrast, both standard and approximate Protectors automatically provide good balance between availability and recovery cost. Their availability is only slightly higher than the target under both schemes (no more than 1 percent of the target). Their costs are low and close to the best results achieved by either the oracle detector or the best possible time-out detector under both schemes (at most 6.3 percent higher than the cost of the oracle detector).

Result 2: Protector achieves the best accuracy. To show the detailed performance of these detectors, we take frequent snapshots over the trace, and look at whether our estimator is accurately predicting the number of remaining replicas for each object. At each snapshot, we compute the fraction of data objects for which the estimator was accurate, underestimated (false positives), or over-estimated the actual replica count (false negatives). Figs. 3b and 4b show the average results obtained at every snapshot under two schemes (with replication and with coding).

Under the replication scheme, standard Protector achieves the highest possible accuracy rate (0.73) of all detectors shown in Fig. 3b. Further, approximate Protector achieves performance very close to that of standard Protector, e.g., approximate Protector's accuracy rate is 0.72, only 1.4 percent lower than that of standard Protector. Since approximate Protector distributes large failure probabilities among peers, it has a relative lower number of false positives and higher number of false negatives. Under the coding scheme, both time-out detectors and Protector achieve lower accuracy rate due to more possible values of remaining replicas. Here, we only show the performance of approximate Protector in that the standard one is not suitable for cases with large t_r. We see that the approximate Protector also outperforms detectors shown in Fig. 4b in terms of accuracy rate.

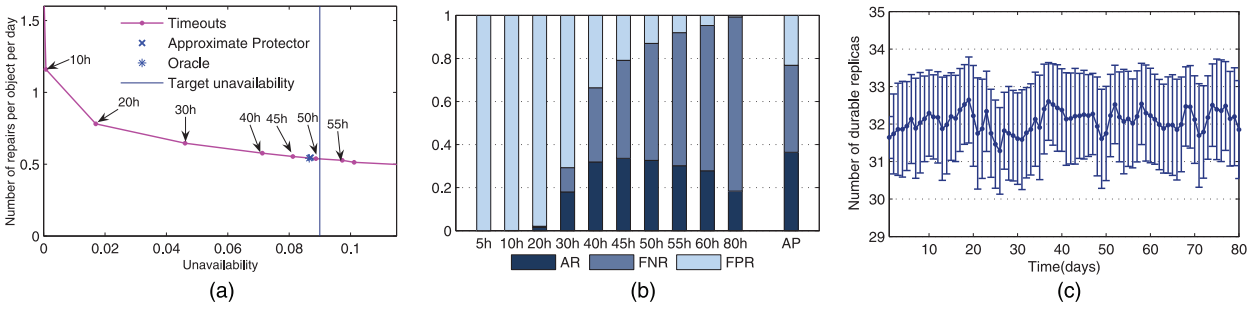

Fig. 4. Performance of standard Protector and approximate Protector with the synthetic trace for the Maze-like environment (under coding scheme). (a) Cost-availability trade-off. (b) Percentage of availability estimates that are accurate, false positives, and false negatives. (c) Replica maintenance with approximate Protector ($b = 6, t_r = 32$).

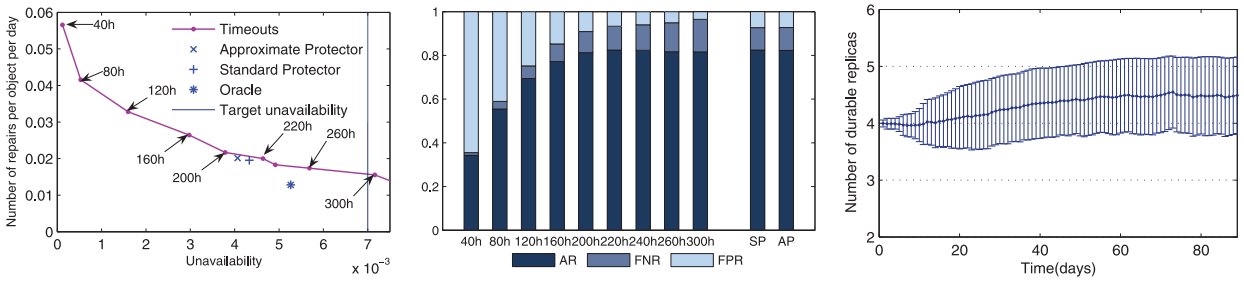

Fig. 5. Performance of standard Protector and approximate Protector with the synthetic trace for the PlanetLab-like environment (under replication scheme). (a) Cost-availability trade-off. (b) Accuracy rate (AR) and false positive/negative rate (FPR/FNR). (c) Replica maintenance with standard Protector ($t_r = 4$).

In addition, we see that time-out detectors that accurately estimate the number of remaining replicas are also those that achieve a well-balanced trade-off between cost and availability (e.g., from 50-hour time-out to 60-hour time-out in Fig. 3b). This confirms our intuition that estimation accuracy is the major factor in deriving a good cost-availability trade-off.

Result 3: Protector maintains the replica count efficiently. Intuitively, it is clear why the Protector is able to derive good trade-offs. With accurate estimates, the system could significantly reduce the fraction of underreplicated objects (due to false negatives) and overreplicated objects (due to false positives). Thus, it maintains the desired number of remaining replicas with a high probability, and leads to a well-balanced trade-off between availability and cost.

To illustrate this point, we look at the remaining number of replicas of each data object when Protector is used, and plot their mean and standard variance at each snapshot in Figs. 3c and 4c. Results clearly demonstrate that Protector is able to correctly maintain the number of replicas over time. With our target replica count at 7, Protector achieves an average of 7.28 replicas per object (4 percent higher than the target), and a standard variance of 0.72 replicas per object. Even when the target replica count jumps to 32 under a coding scheme, Protector maintains an average of 32.1 replicas per object (0.3 percent higher than the target) and a standard variance of 1.2 replicas per object.

Additional remarks. Besides these primary observations, we make some additional remarks on the performance of Protector. Comparing to the oracle detector, Protector provides better availability at a slightly higher cost. This is because Protector still makes false estimates, which the system treats differently. 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 waits for the number to drop. Thus, Protector leans slightly toward more replication and higher availability.

When compared with the time-out-based method, there is a window of time-out values in which time-out detectors meet the availability target while incurring slightly lower cost than Protector. Notice that the existence of such a window is not because the time-outs in it can achieve better accuracy. In fact, as was explained above, Protector leans slightly toward more availability and replication due to different ways of handling false negatives and false positives. Hence, those time-out-based detectors with slight lower accuracy but more false negatives can further reduce the extra availability and replication incurred by Protector, thus forming the window. However, this window is very small (within 5 hours), and the bandwidth saving is not significant (the best time-out detector only saves at most 5.8 percent over Protector in this window). Given the challenge of deriving accurate time-out values, Protector is much more preferred in practice.

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 time-out-based detector cannot use 30-day as the time-out: any time-out larger than 60 hours fails to achieve the availability goal. This again shows that time-out-based detectors require finely tuned time-outs, while Protector can start with very conservative time-outs and still achieve near-optimal availability and cost tradeoff.

Results from PlanetLab in Figs. 5 and 6 show similar trends as the ones in Figs. 3 and 4. Approximate Protector also achieves similar performance to that of standard Protector. Overall, Protector schemes provide availability slightly higher than the target availability and the availability provided by the oracle detector. Given the same availability level, Protector schemes have a smaller recovery cost than the time-out detector. The bandwidth cost of Protector is only slightly higher than that of the oracle detector and that of the best-tuned time-out detector.

4.2 Evaluation Using System Traces

In this section, we use actual system traces collected from a P2P file-sharing system to evaluate the performance of the Protector as compared with time-out-based detectors and the oracle detector. The P2P system that we use is Maze [36], which we briefly describe below.

System environment. With an average of 20K simultaneously online peers, Maze [36] is one of the largest distributed systems on China Education and Research Network (CERNET). Maze is a representative highly dynamic distributed system, because its dynamic level is close to Overnet [6]. 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 by our group. We construct the environment with the system log from 3/1/2005 to 5/31/2005. We use 30-day as the threshold to classify a failure as permanent, which is

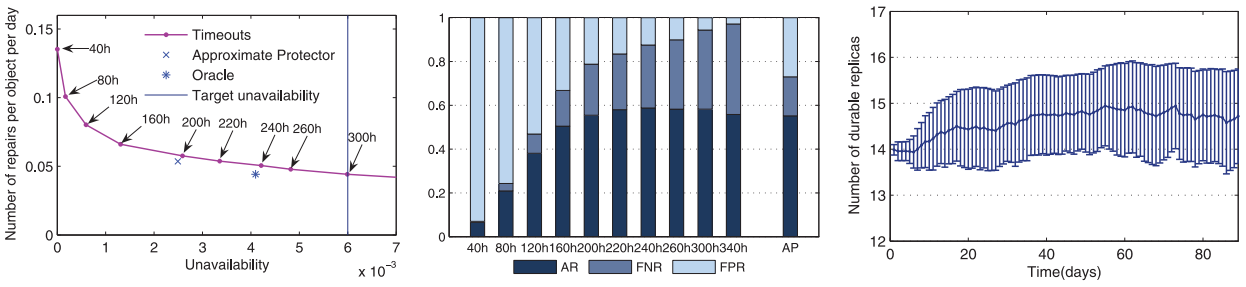

Fig. 6. Performance of standard Protector and approximate Protector with the synthetic trace for the PlanetLab-like environment (under coding scheme). (a) Cost-availability trade-off. (b) Accuracy rate and false positive/negative rate. (c) Replica maintenance with approximate Protector ($b = 6, t_r = 14$).

derived by finding the value T making $P(TTR > T) \leq 0.001$. Fig. 7 plots the complementary cumulative distribution of TTR in Maze system, which clearly shows that the curve is getting flat after 30-day threshold. With this threshold, we use a method similar to [20] and obtain $\mathrm{MLT} = 58$ days.

Workload and simulator. We use 1,000 peers in the system to store 2,000 objects. The historical behavior of all peers in March is used to bootstrap the Protector. In particular, we use the trace data in March with (6) and (5) to estimate $F(d)$. Since the system is relatively stable, we use this measured $F(d)$ to predict behavior of new peers joined in the subsequent months. Given an object and its target number t_r of replicas, the system first places t_r replicas on a group of peers available on 4th April. Then, the simulator added and removed peers 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 peers at that time.

Results. The result of our simulation is summarized in Fig. 8. An important observation is that time-out-based detectors result in higher availability and higher cost, compared with the same time-out values in Fig. 3. The main reason is that the actual distribution of time to failures and time to recovery are not exponential. In particular, the results in [30] 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. Therefore, the time-out-based detectors should increase the threshold appropriately to prevent higher false positive rates and to maintain its best performance.

In contrast to time-out-based detectors that require optimal parameter tuning to achieve good results, Protector maintains optimal performance without any explicit parameter tuning. Using the actual TTR distribution to estimate $F(d)$, (we call this *empirical* $F(d)$), Protector automatically adjusts itself to obtain correct estimates of the number of remaining replicas. Fig. 8c clearly shows that empirical $F(d)$ is in general lower than the $F(d)$ computed based on the exponential distribution assumptions, which we refer to as *exponential* $F(d)$. This means that Protector with empirical $F(d)$ is less aggressive in declaring a failure as permanent and beginning data recovery. Therefore, even when the system behavior does not exactly follow the Markov model, Protector still provides near-optimal trade-off between availability and the recovery cost. Our results show that the recovery cost of approximate Protector is *at most* 14.3 percent higher than the oracle, and 9.1 percent higher than the best-tuned time-out detector under replication and coding schemes (see Figs. 8a and 8b).

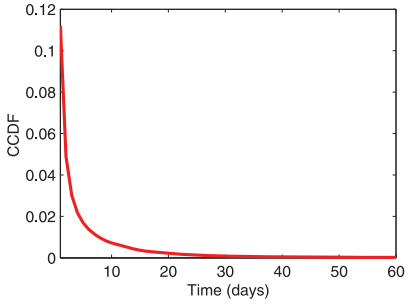

Fig. 7. Complementary cumulative distribution of failure-recovery times.

5 Implementing Protector

In this section, we describe our experience in implementing Protector for storage systems using a structured P2P network or distributed hash tables (DHTs) [19], [28]. Moreover, we deploy a prototype called AmazingStore [1] to validate the real-world implementation and effectiveness of Protector.

5.1 Experience in Highly Dynamic P2P Networks

DHTs organize peers into an ID space and provide a simple primitive for locating a peer. Given a query for a specific key, the associated *lookup* operation can efficiently locate an online peer whose ID is numerically closest to the key [19], [28]. As a result, DHTs have become a general infrastructure for building many P2P storage systems. In a DHT-based storage system, each data object is associated with a "master node" which implements the data maintenance algorithm: the master node monitors the liveness of replicas, adds new replicas according to the data maintenance algorithm (below we center around Protector-based maintenance).

Implementing Protector requires each master node to collect certain types of state information, including explicit metadata on the *replica-to-peer mapping* of each object for which it is responsible, *failure state* for the failure probability function $F(d)$, and *downtime history* of peers hosting the replicas. Each master not only store state information

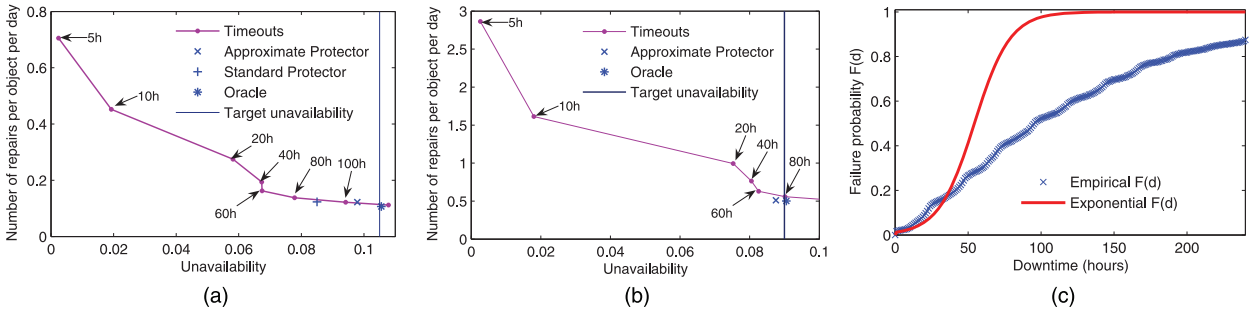

Fig. 8. Performance of Protector with the real trace-driven simulation in the Maze environment. (a) Cost-availability trade-off under replication scheme ($t_r = 7$). (b) Cost-availability trade-off under coding scheme ($b = 6, t_r = 32$). (c) Comparison between empirical $F(d)$ and exponential $F(d)$.

locally, but also continuously backs these data up to successors for available in case it fails.

In general, the master node can be assigned to any peer (e.g., as in TotalRecall [7]), but that may incur large implementation overhead in networks with high churn rates (as is the case with most P2P networks). Collection and synchronization costs may be high in that 1) each peer has to contact many master nodes responsible for different objects it holds, and 2) network dynamics force master nodes to constantly replicate state information to maintain information availability. While developing prototype implementation of Protector in practical P2P environment, we found it necessary to assign master nodes to superpeers (those peers satisfying certain criteria such as high availability, large bandwidth, and computation power), making an intensive usage of the heterogeneous nature of the P2P network.

Therefore, we use a superpeer topology for dynamic P2P networks. Every peer joins the global DHT, and connections between superpeers build up the superpeer DHT, as shown in Fig. 9. Since we can easily construct this topology using DHT algorithms [19], [28], we now focus on the efficient implementation of Protector given this network topology.

The system first partitions peers into different groups (called Storage Clusters), each of which consists of a superpeer (as master) and multiple ordinary peers (as storage hosts). These clusters are self-formed: the system uses each superpeer as a master to initiate a cluster (the superpeer ID is the cluster ID). Each ordinary peer only joins one cluster in either of two ways: First, the master can find ordinary peers to join its cluster, e.g., it first finds a peer by issuing a random lookup in the global DHT, then adds this peer to its cluster if the peer does not yet join a cluster. Second, an ordinary peer can also join a cluster by asking other peers about their masters.

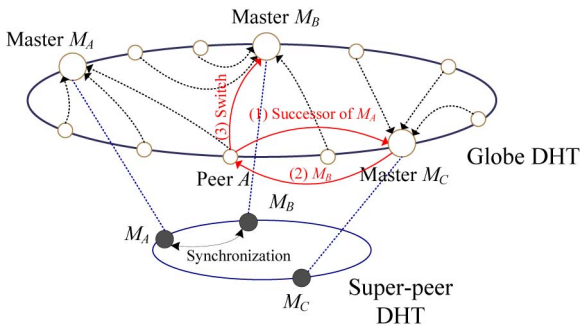

Fig. 9. The cluster-based architecture. Peers are organized into storage clusters, each of which has a master peer (locatable by searching for the master ID in the globe DHT) which monitors and maintains replicas of objects stored in the cluster, and resolves client queries.

Next, the system allows the *write* operation to be performed within each storage cluster. This actually partitions the whole set of data objects over different clusters. In particular, any peer is eligible to receive client[2] write requests. Upon receiving a write request, the peer forwards this request to its master. The master asks a set of peers in the cluster to store object data, and maintains pointers to peers with the object's replicas. By this way, the system assigns the master node of each data object to the master peer of that cluster (also a superpeer), and obtains the replica-to-mapping of each data object.

After that, each master collects failure state and downtime history of peers in its cluster. Each peer announces its presence by periodically (e.g., every 5 minutes) sending heartbeat to its master. The master logs the time when peer leaves and returns (e.g., the start and end of each failure), which are used later to compute the failure probability function $F(d)$. If no heartbeats are received from a particular peer, the master would periodically compute its downtime (that is, how long has this peer been unavailable since its latest online check in). Since failures are usually transient in P2P networks, a peer is excluded from a cluster only when its downtime exceeds a very conservative threshold (e.g., one month).

Finally, the master can maintain objects with Protector. Knowing the replica-to-peer mapping and current peer failure probabilities, the master periodically (e.g., every hour) uses Protector to estimate the number of remaining replicas m for each object stored in the cluster. When m falls below the target t_r, the master repairs $m - t_r$ new replicas randomly among online peers within the cluster, and updates the replica-to-peer mapping.

The superpeer DHT is used to handle master failures. For fault tolerance, each master periodically synchronizes all types of state information with its successors in the superpeer DHT. In the case of a master failure, the closest successor of the failed master in the superpeer DHT will temporarily take over the management of cluster. Fig. 9

2. A client performing a "read" on an object sends a two-part ID. The receiving peer first locates the master using DHT routing on the object's Master ID (MID), then queries the master with the object's Object ID (OID). The master will direct the client to available replicas based on the replica-to-peer mapping of this object.

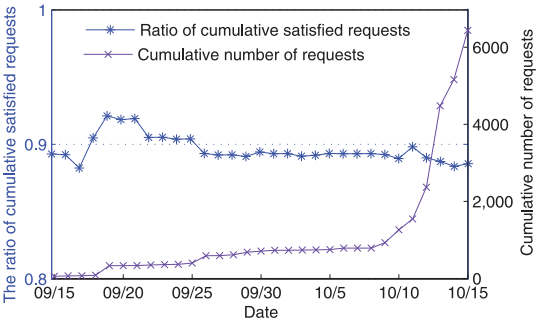

Fig. 10. The service availability of AmazingStore under the real-world request workload (measured from September 15 to October 15, 2009).

illustrates this process. When a peer A detects the failure of its master M_A using periodic probes, it first finds a superpeer M_C by asking other peers about their masters, and then uses M_C as a proxy to locate the closest successor of M_A in the superpeer DHT (e.g., M_B). After that, peer A contacts M_B and takes it as the temporary master.

Notice that each peer only contacts one master and superpeer network is highly stable, so both collection and synchronization costs can be greatly reduced.

5.2 Deployed Prototype and Measurement

We turn the above design choices into a working prototype called AmazingStore [1], a highly available P2P backup and remote storage service. Beyond what the simulation can offer, AmazingStore further validates our implementation and provides us with performance measurements of Protector under real-world workloads.

Users participating in our system provide some of their resources (storage space, upload and download bandwidth) in exchange for using the backup service. A first full prototype of AmazingStore was released for public use on April 3rd, 2009. As of early January 2010, more than 9,820 users had registered accounts. The daily peak of simultaneous online users is above 1,000, and is increasing on a daily basis. In total, the number of objects (compressed archive files or folders) currently stored in the system is 52,055 and continuously rising. The total amount of storage currently occupied in the system is roughly 1.6 TB.

AmazingStore experiences the similar level of peer dynamics as Maze file-sharing system [36] in that they are both deployed over CERNET. Thus, AmazingStore employs a replication scheme with target $t_r = 7$ to guarantee a target availability of 0.9 (like configuration of maze-like system in Section 4), trading imperfect availability for scalability. To validate the effectiveness of our design, we are primarily concerned with two system aspects associated with the Protector, the capacity to deliver the level of availability requested and the efficiency of data replication.

Service availability is the probability that the system is able to satisfy client requests. In Fig. 10, we plot a real-world request load (cumulative number of requests the system received from September 15 to October 15, 2009) and corresponding service availability (the ratio of cumulative satisfied requests to cumulative number of requests). It is encouraging to see that the measured availability was close to our 0.9 availability target (up to 89 percent of the requests was satisfied over the entire month), meaning a well balanced cost-availability trade-off.

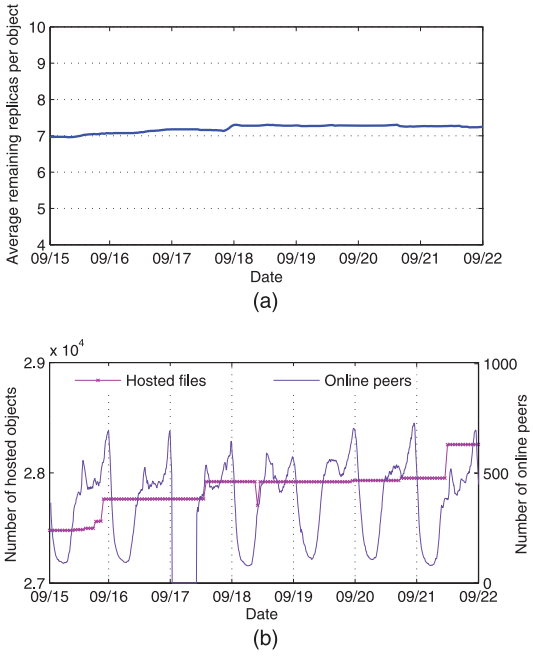

Fig. 11. The evolution of the remaining replicas over a week period (September 15-22, 2009). (a) Average number of remaining replicas per object. (b) Number of objects and online peers along with time (the ticks on the x-axis correspond to midnight (00:00 am) on the days that are labeled).

To demonstrate the efficiency of data replication, we take a closer look at the average number of replicas per object measured during a week stretch of our deployment (September 15-22, 2009), as shown in Fig. 11a. To provide better intuition for system workload and dynamics, we also give the number of objects and online peers in the system over this week period (see Fig. 11b). We see that the system is able to maintain correctly the number of remaining replicas even with substantial peer churn. In particular, with our target replica count at 7, the system achieves an average of 7.19 replicas per object (only 2.7 percent higher than the target). More detailedly, Protector achieves an overall accuracy rate of 65.2 percent (with a false positive rate of 7.5 percent and false negative rate of 27.3 percent) over the entire month (from September 15 to October 15, 2009). Therefore, Protector allows our system to tolerate short-term fluctuations in peer availability, and to maintain a constant number of live data replicas.

To examine another property of our system, durability, we also show the measured probability density function (PDF) of the number of remaining replicas per object in Fig. 12. We see that the distribution resembles a Gaussian and the replica count clusters around the target ($t_r = 7$). Hence, only 0.06 percent of the objects were lost over an approximate six-month deployment period (from April 4 to September 22, 2009). Notice that the participation of peers lasts only a period of two months on average. Our system actually provides high data durability for each peer over its participation period, although Protector has a relative large false negatives. As mentioned before, the reason for this lies in two points: 1) the omitted repair actions (due to false negatives) could be compensated quickly in the following estimate interval to avoid further decrease in the number of

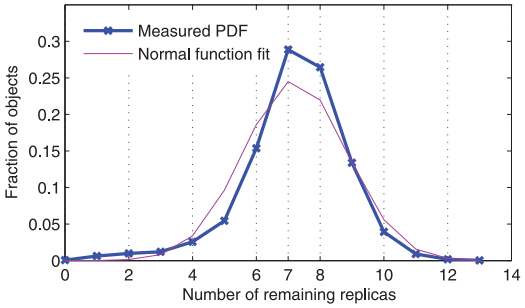

Fig. 12. A snapshot of PDF of the number of remaining replicas (taken at midnight on 22 September 2009).

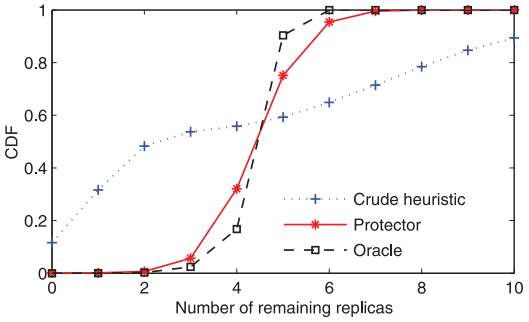

Fig. 13. CDF of the number of replicas per object at the end of the trace.

remaining replicas; 2) additional repair actions (due to false positives) bring extra durability, further compensate the false negatives. Thus, durability can also be guaranteed with Protector in our system.

6 Extended Use Case

Existing replication maintenance strategies can be classified as reactive maintenance or proactive maintenance. Reactive maintenance is able to replace lost replicates after failures at the expense of a bursty bandwidth costs. In contrast, proactive maintenance evens out burstiness in maintenance traffic by constantly replicating replicas as necessary.

While the previous sections focused on demonstrating Protector's usefulness in the specific context of reactive maintenance, this section briefly shows that proactive maintenance can also adopt Protector for fine-gained replication scheduling.

6.1 Protector-Based Proactive Maintenance

Protector can be used in proactive maintenance systems such as Tempo [27]. Each peer in Tempo sets an outgoing bandwidth cap b_i that appropriately limits the total number of bytes sent per unit time. The peers cooperate and attempt to maximize *durability* by constantly creating new replicas with bandwidth b_i, whether or not they are needed at the moment.

Since the proactive system operates constantly in the background, the bandwidth cap b_i should be low enough to prevent network congestion and storage overflow, usually around several kilobytes per second. However, it may take a long time to produce a new replica. As a result, systems like Tempo must prioritize the replication of different objects based on how critical each object needs additional replicas. Tempo puts objects with fewer number of *available* replicas at high priority. However, this method overlooks the fact that many departed peers can rejoin the network and bring back their stored replicas. Thus, it can only provide an approximate estimate of how badly each object needs replicas.

Intuitively, it makes sense to give priority to objects with fewer remaining replicas, since they are more likely to be destroyed by unpredictable future failures. Therefore, Protector is particularly useful here, since the problem of assigning replication priority to an object is directly tied to the number of remaining replicas each object has. In general, we can deem a priority assignment policy as a monotonically decreasing function $P(m)$, where m is the estimate of remaining replicas given by Protector.

Given a subset of data objects S associated with the same master, our method can be applied as follows: For each object i in S, the master periodically updates m_i based on the current downtime of individual peers and the Protector algorithm. Whenever a peer is idle (e.g., it has excess storage and bandwidth resources), the master would tell it to download a replica of the object with the lowest value of m. Like reactive maintenance, we can implement proposed proactive maintenance using the architecture showed in Fig. 9.

6.2 Trace-Driven Evaluation

In this section, we present a simulation-based study that measures the impact of our replication prioritization scheme on durability. We first introduce our simulator and then present simulation results and discuss their implications.

We construct the simulation environment with the Maze log from March 1 to June 30, 2005. We exclude some extremely unstable peers by only using those that have appeared at least two times. We simulate 1,000 peers maintaining 10,000 20 MB objects. In our simulation, we set the recovery bandwidth cap b to 1 KB/s (including both upload and download bandwidth) and storage cap to 1 GB at each peer.

The simulations in this section proceed as follows: peers join and leave the system, as dictated by the observed time in the trace. We uses the first month trace to obtain the parameters needed by the Protector. Then, objects are randomly scattered across peers, and each with three replicas. Once an idle peer is detected, the master finds the object with the lowest replication level based on the Protector, and asks the peer to download this file with a bandwidth b.

To evaluate our Protector-based method, we compare it against the following two methods: 1) a crude heuristic that only uses the number of available replicas as an indicator of object's replication level; 2) an oracle heuristic which knows the exact number of replicas. The durability can be easily evaluated through looking at the cumulative distribution of the number of remaining replicas per object at the end of the trace, as shown in Fig. 13.

From the figure, we see that the crude heuristic is quite inefficient in a highly dynamic system. The result shows that about 12 percent of the objects are lost and over 46 percent of the objects are underreplicated (e.g., less than 3 remaining replicas) at the end of trace. On the other hand, there is a subset of objects that achieve much greater replication level than the average (e.g., over 10 percent of the objects have more than 10 replicas). This is because the

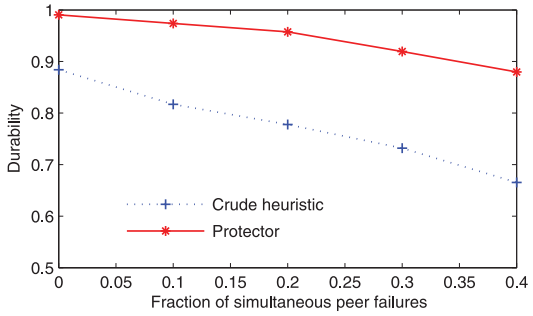

Fig. 14. Resistance to correlated peer failures.

crude heuristic cannot find objects with fewer replicas due to diurnal correlation between peers. Thus, many objects have not been repaired and become underreplicated. Over time, objects with few replicas lose the competition for excess storage since they are often unavailable when there is idle bandwidth.

In contrast, we observe that accurately putting rare objects at high replication priority enables the system to equalize the replication level of all objects and thus improve the overall data durability. As Protector provides a more accurate estimate of the rarity of each object, only about 0.1 percent of files are lost at the end of trace. This means that the system has little trouble in achieving high durability, and also clearly demonstrates the usefulness of Protector. Further, by comparing the curves in Fig. 13, we observe that the performance of Protector is only slightly below that of the ideal oracle heuristic method.

Moreover, the fraction of underreplicated objects is reduced to 3.5 percent after adopting the Protector. This enables the system to provide a more durable storage service, especially in the case of severe machine failure correlations (e.g., disaster events). To confirm this benefit, we remove a certain fraction of peers in the middle of the trace to simulate a large correlated failure event. This burst of permanent failures tend to destroy objects that only have a few remaining replicas. Fig. 14 shows the durability at the end of trace as a function of the fraction of the peers removed. We see that our method produces a slower decrease in durability as the number of simultaneous failures increases. This suggests that Protector makes the system more resilient to large correlated failure events.

7 Conclusion

Detecting permanent failures is the key to guaranteeing high data availability while minimizing system maintenance bandwidth in P2P storage systems. We have presented a novel methodology that estimates the number of remaining replicas rather than determining if a single failure is permanent or transient.

The proposed Protector system is based on 1) leveraging prior failure statistics, and 2) making estimates across a group of replicas which balance false positives for some peers against false negatives for others, thus providing high estimation accuracy. We have proven that Protector provides the best estimate on the number of remaining replicas in the system among all methods including the time-out-based methods. Moreover, we show how to tune Protector based on history without setting manual parameters. We demonstrate that Protector enables the system to maintain objects in the most cost-efficient manner through an extensive simulation-based evaluation. Both model-based and trace-driven simulations verify that Protector achieves performance close to that of the perfect oracle detector and outperforms most time-out-based detectors.

Further, we present our experience in implementing Protector in low-availability P2P environments. To verify the actual implementation, we have built and deployed a P2P storage system with the Protector called AmazingStore. AmazingStore has been running for eight months, and the daily peak of simultaneous online users is above 1,000. Experience with this system shows Protector is a useful method that enables efficient long-term data maintenance.

Acknowledgments

This work is supported by the National Basic Research Program of China under Grant No. 2011CB302305 and the National Natural Science Foundation of China under Grant No. 60873051 and No. 61073015.

References

[1] Amazingstore, http://en.amazingstore.org/, 2010.
[2] Cleversafe Inc., http://www.cleversafe.org/dispersed-storage, 2011.
[3] Wuala, Peer-to-peer storage system, http://wua.la/en/home.html, 2010.
[4] M. Bertier, O. Marin, and P. Sens, "Implementation and Performance Evaluation of an Adaptable Failure Detector," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '02)*, 2002.
[5] M. Bertier, O. Marin, and P. Sens, "Performance Analysis of a Hierarchical Failure Detector," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '03)*, 2003.
[6] R. Bhagwan, S. Savage, and G. Voelker, "Understanding Availability," *Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '03)*, 2003.
[7] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G.M. Voelker, "Total Recall: System Support for Automated Availability Management," *Proc. Symp. Networked Systems Design and Implementation (NSDI '04)*, 2004.
[8] C. Blake and R. Rodrigues, "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two," *Proc. Ninth Workshop Hot Topics in Operating Systems (HotOS '03)*, 2003.
[9] A. Bondavalli, S. Chiaradonna, F.D. Giandomenico, and F. Grandoni, "Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults," *IEEE Trans. Computers*, vol. 49, no. 3, pp. 230-245, Mar. 2000.
[10] A. Casimiro, P. Lollini, M. Dixit, A. Bondavalli, and P. Verissimo, "A Framework for Dependable QoS Adaptation in Probabilistic Environments," *Proc. 23rd Ann. ACM Symp. Applied Computing (SAC '08)*, 2008.
[11] W. Chen, S. Toueg, and M.K. Aguilera, "On the Quality of Service of Failure Detectors," *IEEE Trans. Computers*, vol. 51, no. 5, pp. 561-580, May 2002.
[12] B. 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. Third Symp. Networked Systems Design and Implementation (NSDI '06)*, 2006.
[13] L.P. Cox and B.D. Noble, "Samsara: Honor Among Thieves in Peer-to-Peer Storage," *Proc. ACM Symp. Operating Systems Principles (SOSP '03)*, 2003.
[14] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01)*, 2001.
[15] A. Datta and K. Aberer, "Internet-Scale Storage Systems under Churn—A Study of the Steady State Using Markov Models," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '06)*, 2006.

[16] L. Falai and A. Bondavalli, "Experimental Evaluation of the QoS of Failure Detectors on Wide Area Network," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '05),* 2005.
[17] A. Gharaibeh and M. Ripeanu, "Exploring Data Reliability Tradeoffs in Replicated Storage Systems," *Proc. 18th ACM Int'l Symp. High Performance Distributed Computing (HPDC '09),* 2009.
[18] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures," *Proc. Second Symp. Networked Systems Design and Implementation (NSDI '05),* 2005.
[19] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* 2002.
[20] L. Ni and A. Harwood, "A Comparative Study on Peer-to-Peer Failure Rate Estimation," *Proc. Parallel and Distributed Systems (ICPADS '07),* 2007.
[21] 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. Int'l Conf. Dependable Systems and Networks (DSN '04),* 2004.
[22] V. Ramasubramaniana and E. Sirer, "Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays," *Proc. Symp. Networked Systems Design and Implementation (NSDI '04),* 2004.
[23] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: The Oceanstore Prototype," *Proc. USENIX Conf. File and Storage Technologies (FAST '03),* 2003.
[24] R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication," *Proc. Fourth Int'l Workshop Peer-to-Peer Systems (IPTPS '05),* 2005.
[25] S. Ross, *Stochastic Processes.* Wiley, 1996.
[26] Seagate, "Laying it s.m.a.r.t. Self Monitoring, Analysis and Reporting Technology (s.m.a.r.t) Frequently Asked Questions," http://www.seagate.com/support/kb/disc/smart.html, 2010.
[27] E. Sit, A. Haeberlen, F. Dabek, B. Chun, H. Weatherspoon, R. Morris, M.F. Kaashoek, and J. Kubiatowicz, "Proactive Replication for Data Durability," *Proc. Fifth Int'l Workshop Peer-to-Peer Systems (IPTPS '06),* 2006.
[28] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service or Internet Applications," *Proc. ACM SIGCOMM,* 2001.
[29] K. Tati and G. Voelker, "On Object Maintenance in Peer-to-Peer Systems," *Proc. Fifth Int'l Workshop Peer-to-Peer Systems (IPTPS '06),* 2006.
[30] J. Tian and Y. Dai, "Understanding the Dynamic of Peer-to-Peer Systems," *Proc. Sixth Int'l Workshop Peer-to-Peer Systems (IPTPS '07),* 2007.
[31] J. Tian, Z. Yang, W. Chen, B.Y. Zhao, and Y. Dai, "Probabilistic Failure Detection for Efficient Distributed Storage Maintenance," *Proc. 27th IEEE Int'l Symp. Reliable Distributed Systems (SRDS '08),* 2008.
[32] J. Tian, Z. Yang, and Y. Dai, "A Data Placement Scheme with Time-Related Model for P2P Storages," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '07),* 2007.
[33] D.N. Tran, F. Chiang, and J. Li, "Friendstore: Cooperative Online Backup Using Trusted Nodes," *Proc. First Workshop Social. Network Systems (SocialNets '08),* 2008.
[34] H. Weatherspoon, B.G. Chun, C. So, and J. Kubiatowicz, "Long-Term Data Maintenance in Wide-Area Storage Systems: A Quantitative Approach," *Computer,* 2005.
[35] H. Weatherspoon, T. Moscovitz, and J. Kubiatowiczz, "Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems," *Proc. Int'l Workshop Reliable Peer-to-Peer Distributed Systems (RPPDS '02),* 2002.
[36] M. Yang, B.Y. Zhao, Y. Dai, and Z. Zhang, "Deployment of a Large Scale Peer-to-Peer Social Network," *Proc. First Workshop Real, Large Distributed Systems (WORLDS '04),* 2004.

Zhi Yang received the BS degree from Harbin Institute of Technology in 2005 and the PhD degree in computer science from Peking University in 2010. He is now a postdoctoral researcher at Peking University, collaborating with Professor Yafei Dai. His research interests include peer-to-peer (P2P) systems, as well as distributed storage systems. He has focused on P2P storage, dealing with the churn problem, data availability and reliability model.

Jing Tian received the BS and PhD degrees in computer science from Peking University in 2002 and 2007, respectively. He is a recipient of the Peking University Excellent Graduate Student Award. His main research interests include the reliability, the availability, and the security of the Peer-to-Peer storage system. He is one of the founders and the second chair of YOCSEF-GS (China Computer Federation Young Computer Scientists & Engineers Forum for Graduate Students). Since 2004, he has been a founding team member of a startup company Kuwo, and he is currently the director of Engineer. Kuwo's production is an online Peer-to-Peer streaming music player, which is one of the most popular software in China now.

Ben Y. Zhao received the BS degree from Yale University in 1997 and the MS and PhD degrees in computer science at U.C. Berkeley in 2004. He is currently an associate professor at the Computer Science Department, U.C. Santa Barbara. He is a recipient of the National Science Foundation's CAREER award, MIT Technology Review's TR-35 award (Young Innovators Under 35), and ComputerWorld Magazine's Top 40 Technology Innovators award. His research Interests include the areas of security and privacy, networked and distributed systems, wireless networks, and data-intensive computing. Most recently, he has spent time on measuring, analyzing, and modeling online social networks, as well as studying systems issues in dynamic spectrum access networks.

Wei Chen received the bachelor's and master's degrees from the Department of Computer Science and Technology, Tsinghua University, and the PhD degree from the Department of Computer Science, Cornell University. He is now a lead researcher in the Theory group of Microsoft Research Asia. His main research interests include theory of distributed computing, fault tolerance, and algorithmic aspects of game theory and social networks. He is a winner of the 2000 William C. Carter award at 2000 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), because of the paper "On the quality of service of failure detectors" based on his PhD dissertation.

Yafei Dai received the PhD degree in computer science at Harbin Institute of Technology. She is currently a professor at the Computer Science Department, Peking University. Her research areas include networked and distributed systems, P2P computing, network storage, and online social networks. She is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.