Exploring the Cost-Availability Tradeoff in P2P Storage Systems

Zhi Yang, Yafei Dai, Zhen Xiao
Department of Computer Science, Peking University
Beijing, China
yangzhi@net.pku.edu.cn, {dyf, xiaozhen}@ pku.edu.cn

Abstract—**P2P storage systems use replication to provide a certain level of availability. While the system must generate new replicas to replace replicas lost to permanent failures, it can save significant replication cost by not replicating following transient failures. However, in real systems, it is impossible to reliably distinguish permanent and transients failures, resulting in a tradeoff between high recovery cost and low data availability. In this paper, we analyze the use of timeouts as a mechanism to navigate this tradeoff. We address the challenging problem of how to choose a timeout to walk the fine line between causing unnecessary replication due to detection inaccuracy, and reducing availability due to detection delay. We conduct simulations based both on synthetic and real traces, and show that the performance of our selected timeout closely approximates the optimal performance that can be achieved by timeouts, and even that of an "oracle" failure detector.**

Keywords-P2P storage; availability; timeout-based detectors

I. INTRODUCTION

Peer-to-peer (P2P) storage is an important class of distributed systems which can aggregate storage and bandwidth of a large number of personal computers to provide service for various applications, such as scientific data library [1] and P2P-VoD systems [2]. Due to the prevalence of peer dynamics (*i.e.* churn), a fundamental issue in P2P storage systems is how to tolerate failures and departures of peers [3, 4]. To meet the service level agreements (SLA), a system must achieve a certain level of availability for the data stored in it. To do so, current storage systems [5-9] usually replicate data object across multiple peers so that even if some replicas become unavailable due to their host failures, the object is still available by accessing other online replicas. Furthermore, to maintain long-term availability, a system must also constantly regenerate new replicas to compensate lost ones due to peer failures [5-9]. However, maintaining availability also incurs a heavy cost on storage systems since replica generation consumes a large amount of bandwidth [3, 4]. Clearly, high bandwidth cost can congest the network and significantly reduce system throughput, while reducing the number of replicas generated might result in an unacceptable level of data availability. Therefore, the challenge in building today's storage systems is to carefully navigate this cost-availability tradeoff, by reducing maintenance cost as much as possible while maintaining the desired level of availability [3, 6, 8].

This challenge is further complicated by the fact that peer failure can be either transient or permanent. A peer suffering a transient failure can eventually rejoin the network, bringing back its stored data. Clearly, permanent failures must be addressed by restoring the level of data replication after a peer has permanently failed. However, an object with sufficient replicas can tolerate transient failures without sacrificing data availability. Therefore, it is desirable that an efficient storage system would reliably distinguish between peers that are temporarily down and those have permanently failed, and only pay the costs of replica generation following permanent failures. Unfortunately, reliably distinguishing permanent and transient failures turns out to be a daunting task, because peers are unresponsive in both cases [4, 8, 9].

Timeouts have proven to be an effective way to distinguish failures. The basic idea is to allow the system to wait for some preset timeout threshold before identifying a failure as permanent. Admittedly, this information is not always correct or up-to-date. In practice, a key barrier to adopting timeout is how to determine a suitable threshold [9]. Choosing the timeout threshold walks a fine line between high values that incur large detection delay and reduce availability, and low values that produce false positives and unnecessary replica generation. Thus, ***how to select timeouts to navigate cost-availability tradeoffs remains unanswered***.

To explain our work more precisely, we address the important differences between early timeout-based detectors (e.g. [10-12]) and new detectors we have designed. First, the objectives of the failure detectors are different. While early failure detectors focused on providing fast and accurate detection of all failure events (transient and permanent), our new failure detectors are designed specifically to distinguish permanent from transient failures. Second, while early failure detectors mainly relied on message delay distributions to adjust the QoS of the detector itself, new failure detectors rely on the distribution of permanent failures vs. transient failures to navigate the tradeoff between data availability and maintenance cost at system level.

Contributions

- We apply a semi-Markov process (SMP) to predict peer failures, and establish the first model for systems that adopt memory-based repair strategy (e.g. the system reintegrates the returning replicas).
- We propose a timeout selection mechanism to best achieve target level of availability, i.e., to maintain target availability with minimal replication cost. The key idea is to allow the increased availability due to false positives to exactly compensate for the reduced availability due to detection delay.
- To limit the search space and to reduce the computation cost, we propose an approximate solution to the timeout selection problem by pruning rare events. Through exten-

sive simulations using both synthetic and real traces, we show that not only does our selected timeout provide a near-optimal tradeoff between cost and availability; it also approximates the performance of the oracle failure detector.

This paper proceeds as follows: In Section II, we describe the basic idea of timeout tuning. In Section III, we show the tuning method in detail, and present some attractive features of this method. In Section IV, we evaluate the performance of our approach using both synthetic traces and real trace driven simulations. We present related work in Section V, and conclude the paper in Section VI.

II. Availability Maintenance Strategy

Replication is used to provide high data availability in a statistical sense. Under the assumption of independence, if the availability of an individual peer in the system is p_c, the availability of a data object with r replicas (each on a different peer) is $1-(1-p_c)^r$. Therefore, existing storage systems maintain a target replication level t_r for each data object to guarantee its availability. To minimize replication cost, t_r is usually defined as the minimum r making object availability achieve target p_t (as in [4, 6, 8, 9]), which means:

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

However, to significantly extend the availability of an object for periods exceeding individual peer lifetimes, the system must compensate for lost replication by creating new replicas, where timeouts play a key role by avoiding misclassifying transient failures as permanent failures. To eliminate the effect of parameter t_r on timeout selection, we consider an object with a target replication level of t_r as t_r duplicate objects, each with target replication level one. Thus, the replication level of an object is maintained as follows:

For a given object o, the system first gives numbers to its t_r replicas in the initialization stage, calling them $r_1, \ldots, r_{tr}$. Then, it periodically detects the current number of non-timeout peers hosting r_i ($1 \le i \le t_r$); if this number $m_i = 0$, the system generates a new replica numbered as r_i on a randomly selected peer; if $m_i > 0$, no actions are necessary. In fact, the system considers a numbered replica as the maintenance unit, whose target replication level is fixed at one. By this way, we can restrict our attention on a numbered replica and use the same timeout for various availability requirements.

Let p_r be the availability of a numbered replica r_i, (*e.g.*, the fraction of time that the replica r_i is available during the maintenance process), then the *long-term* availability of an object can be rewritten as $1-(1-p_r)^{tr}$. Clearly, increasing timeout can reduce replication cost by reducing false positives, while this also decreases availability for incurring a larger detection delay. Thus, given that t_r is computed by equation (1), ***the best timeout is the value making $p_r = p_c$***, where the optimal cost-availability tradeoff is achieved since replication cost is at its minimum possible value with the availability target satisfied. ***In other words, the best timeout is the value that allows the increased availability due to false positives to exactly compensate for the reduced availability due to detection delay.***

The advantage of our selection method is that it can be performed on a per-replica basis. Performing selection across the group of replicas is difficult since parameter t_r complicates the underlying model [3, 9]. To our knowledge, no one has answered how to select timeouts in such a way. Further, our simulation study shows that these two selection methods essentially achieve similar tradeoffs; though the corresponding timeouts are usually different. We refer the readers to our technique report [13] for details on this comparison.

III. Adaptive Timeout Selection

For simplicity, we first show how to compute the timeout by assuming that the probabilistic behavior of peers does not change over time, and repair time is reasonably small. We then drop these assumptions and suggest ways to modify the algorithm so that it dynamically adapts to changes in peer behavior and eliminates the effect of repair time.

A. Peer Failure Model

To keep the derivations tractable, we impose the following restrictions on the system. We first make a conventional assumption that peer lifetimes are exponentially distributed with mean T. This has been shown to be valid in certain P2P environments composed of servers (e.g. PlanetLab [8, 9, 14]). Below, we also comment on the appropriateness of this assumption in the cases with millions of home users (e.g. the Maze system [15]). In addition, we assume that peer lifetimes are notably longer than the peer downtimes (one contiguous interval of time when a peer is unavailable), that is, $T \gg \mu$, where μ is the average downtime. This assumption is necessary to guarantee the existence of an efficient timeout [9], and is usually satisfied in typical P2P environments [8, 15, 16]. Finally, we assume that peers are independent [17].

The above assumptions allow us to model peer behavior by a continuous semi-Markov chain [18], where a peer always begins its existence in *online* state, alternates between *online* state and *offline* state during its life, and finally enters *dead* state (for simplicity, from now on we use *live* peer to refer a peer that has not yet entered the dead state). Contrary to a standard Markov chain (e.g. used in [19, 20]), 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*.

Figure 1 depicts the embedded Markov chain of the peer behavior, where p is the transition probability from online state to dead state. Clearly, peer lifetime is the absorption time of the chain in dead state. If the peer generates N number of transient failures before it finally permanently fails (that is, the chain visits offline states N times before it reaches dead state), we have

$$T = E[N](\lambda + \mu) + \lambda \tag{2}$$

where λ denotes the mean session duration (one contiguous

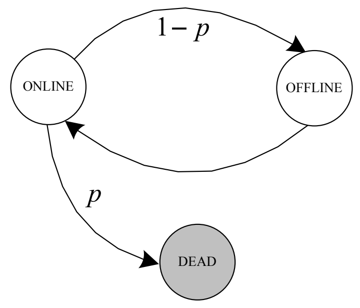

Figure 1. The embedded Markov chain model of a peer life.

interval of time when a peer is available).

Notice that N follows a geometric distribution, hence, $E[N] = (1-p) / p$. Solving equation (2) and using $T \gg \mu$ to simplify yields

$$p = (\lambda + \mu)/T \tag{3}$$

It easy to get that peer availability $p_c = \lambda / (\lambda+\mu)$, i.e., the probability that a peer is online during lifetime.

B. Replica Availability Model

The above peer failure model must be translated in an availability model for analyzing the impact of timeout on long-term replica availability p_r. We do so by assuming that the *object lifetime* is long enough to overcome any transient effects and allow asymptotic results from renewal process theory to hold. This assumption is usually satisfied in practice since a combination of replication and repair mechanism significantly extends the object lifetime for periods exceeding individual peer lifetime. Let $n_i(t)$ denote the number of live peers hosting the numbered replica r_i at time t. Since the peer lifetime is exponential, we see that the stochastic process $n_i(t)$ is a regenerative process with the regeneration point defined at a replacement epoch before which there are no live peers hosting the replica r_i (e.g. $n_i(t)=0$) [19].

In Figure 2, we illustrate the sample path of a regeneration cycle, say X_1. We observe that the process makes an upward transition (a repair event) when all peers simultaneously time out, or alternatively, makes a downward transition when a permanent failure occurs. Suppose that n is the maximum number of live peers during the maintenance process. Let T_k be the amount of time in a cycle with k peers alive, then we have $X_1 = T_0+T_1+T_2+\ldots T_n$. Let p_k denote the long-run average fraction of time that $n_i(t) = k$.

To find the best timeout, we concentrate on modeling the average availability of a numbered replica. A simple application of the law of total probability gives the average replica availability:

$$p_r = \sum_{k=0}^{n}\left[1-(1-p_c)^k\right]p_k \tag{4}$$

Applying the theorem of regenerative process [19], we have:

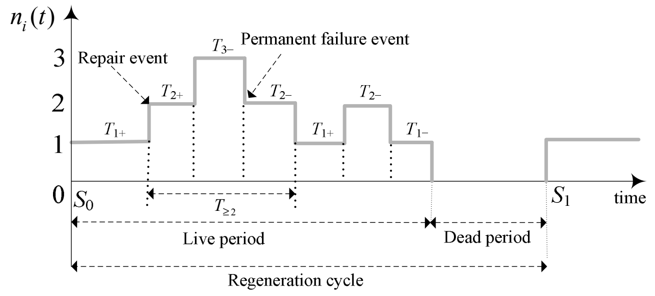

Figure 2. Regenerative process $n_i(t)$ depicting the evolution of the number of live peers hosting the numbered replica r_i.

$$p_k = \lim_{t\to\infty}\frac{\int_0^t P\{n_i(t)=k\}}{t} = \frac{E[T_k]}{E[X_1]} \tag{5}$$

To derive $E[T_k]$ and $E[X_1]$ respectively, we first partition a regenerative cycle into intervals marked by the transition epochs (see Figure 2). For intervals with k peers alive, define T_{k+} as the length of intervals after which a repair happens, and T_{k-} as the length of interval after which a permanent failure happens. Let N_{k+} and N_{k-} denote the number of variables T_{k+} and T_{k-} in a cycle respectively (e.g. the figure shows that $N_{1+}=2$, $N_{1-}=1$), then using Wald's equation [19], we have

$$E[T_k] = E[N_{k-}](q_k E[T_{k+}] + E[T_{k-}]) \tag{6}$$

where $q_k = E[N_{k+}] / E[N_{k-}]$. Clearly, q_k reflects the average number of upward transitions from state k to k+1 before a downward transition from state k to k–1.

To compute $E[T_k]$ in a recursive way, we let $T_{\geq k}$ denote one continuous period of time when the process stays at a state with at least k peers alive. For example, the figure shows that $T_{\geq 2} = T_{2+}+T_{3-}+T_{2-}$. Notice that $E[N_{1-}] = 1$ since each regeneration cycle has only one dead period, then, we obtain

$$\begin{cases} E[T_{\geq 0}] = E[T \geq 1] + E[T_0] \\ E[T_{\geq k}] = q_k(E[T_{k+}] + E[T_{\geq k+1}]) + E[T_{k-}] \; (0<k<n) \\ E[T_{\geq n}] = E[T_{n-}] \end{cases} \tag{7}$$

We note here that $T_{\geq 0}$ is also the length of a regenerative cycle X_1. Define $q_0 = 1$, we can drive the expected value of regenerative cycle:

$$E[X_1] = E[T_0] + \sum_{k=1}^{n-1}\left[\left(\prod_{j=0}^{j=k-1} q_j\right)\left(q_k E[T_{k+}] + E[T_{k-}]\right)\right] + \left(\prod_{j=1}^{n} q_j\right)E[T_{n-}] \tag{8}$$

and the expected value of T_k is given by

$$E[T_k] = \left(\prod_{j=0}^{j=k-1} q_j\right)\left(q_k E[T_{k+}] + E[T_{k-}]\right) \quad (0<k<n) \tag{9}$$

C. Approximate Tuning Approach

Clearly, the probability of finding $n(t) = k$ decreases fast with increasing k in that more peers are less likely to time out simultaneously. To avoid incurring high computational cost, we try to obtain approximate solutions by pruning rare states. By an approximate solution, we mean that it yields a timeout which makes $p_r \approx p_c$ (that is, a near perfect counterbalance). Thus, the true replica availability $p_r = p_c + \varepsilon$ where ε is the offset (or absolute error). To make sure the target availability is satisfied; we shall be interested only in conservative approximations with $\varepsilon > 0$. Since the replication cost monotonically decreases to its possible minimum value c_{min} as $p_r \to p_c$, it is more preferable to indicate the quality of approximation solution by the *offset relative to* p_c (e.g., ε / p_c). We avoid using the offset between actual replication cost and c_{min} since it is difficult to know the exact value of c_{min}.

To obtain an approximation solution, we first need to get an approximation of the replica availability p_r, and then derive a timeout by equaling the approximate replica availability p_r* to the peer availability p_c. By this way, we get a timeout which makes $p_r \approx p_c$ in that $p_r \approx p_r$* and p_r* = p_c. What we need to determine, essentially, is how many number of states are retained to get p_r*. Let n_0 denote the number of retained states, then, states $n_i(t) > n_0$ will be ignored. It is easy to verify that $n_0 \geq 2$; otherwise, the availability reduced in state $n_i(t) = 0$ cannot be compensated. To make our approximation simplest, we retain a minimum number of states and thus $n_0 = 2$. That is, we only consider the increased availability produced by one extra peer. Below, we show some attractive features of our approximation.

Given that $n_0 = 2$, we discard terms with k > 2 in (8) and (4), and get the approximations of the average length of regenerative cycle and replica availability:

$$\begin{cases} E[X_1]^* = E[T_0] + \left(q_1 E[T_{1+}] + E[T_{1-}]\right) + q_1 E[T_{2-}] \\ p_r^* = \dfrac{p_c}{E[X_1]^*}\left[E[T_{1-}] + q_1 E[T_{1+}] + [1+(1-p_c)]q_1 E[T_{2-}]\right] \end{cases} \quad (10)$$

Equating p_r* to peer availability p_c, and simplifying the equation, we obtain

$$E[T_0] = (1-p_c) q_1 E[T_{2-}] \quad (11)$$

The term on the left side of equation (11) gives the decreased availability due to detection delay, while the term on the right side gives the increased availability due to false positives. The equation shows the timeout is selected to enable them compensate for each other.

D. Timeout Estimation

Now, we examine each variable in (11). First, $E[T_0]$ is the average detection delay and hence is given by the timeout δ ($E[T_0] = \delta$). Next, for $k > 0$, T_{k-} is given by the minimum of k peer residual lifetimes. Given exponential lifetimes, T_{k-} is yet another exponential random variable with mean $E[T_{k-}] = T / k$. Finally, notice that a repair happens at state k only when k peers *simultaneously* time out before any of them permanently fails. In the case of k =1, a peer generates $(1-p) / p$ number of downtimes on average, before it permanently fails (see subsection III.A), and each downtime d times out with a probability of $P(d \geq \delta)$. Applying the mean of binomial distribution, we get $q_1 = F_c(\delta)(1-p) / p$ where $F_c(x)$ is CCDF (Complementary Cumulative Distribution Function) of downtime. Substituting these results into equation (11), we get:

$$(1-p_c)\frac{1-p}{p}F_c(\delta)\frac{T}{2} = \delta \quad (12)$$

Notice that $(1-p_c)\, T = \mu / p$, we can rewrite (12) as the following non-linear equation:

$$\frac{(1-p)\mu}{2p^2}F_c(\delta) - \delta = 0 \quad (13)$$

For the case of exponential downtime distribution ($F_c(x) = e^{-x/\mu}$), we can obtain a closed-form solution of equation (13):

$$\delta = \left(1 - \ln\left((1+c)\sigma\right)/(1+\sigma)\right)\sigma\mu \quad (14)$$

where $c = 2p^2/(1-p)$ and $\sigma = \ln(1+1/c)$.

However, solution (14) also provides a useful conservative estimate for cases of heavy-tailed downtime distributions, which include Pareto, lognormal, Weibull and Cauchy distributions. This formally follows the fact that heavy-tailed downtimes imply a larger fraction of long downtime durations, thus δ should be increased to further reduce false positives. To get better solutions, one can consider using the empirical downtime distribution measured from system trace, and solves the non-linear equation (13) with existing methods (such as the bisection method).

E. Quality Analysis

We first give a conservative bound on q_k (refer to the appendix A to [13] for a detailed derivation).

Proposition 1: *For an approximation retaining* n_0 = 2 *states, parameter* q_k *satisfies*,

$$q_k < 2^{k/2} f^{k-1} \quad (15)$$

where $f = p / (1-p)$.

Using this inequality, we get a bound on the offset relative to p_c (refer to the appendix B to [13] for a detailed derivation).

Proposition 2: *For an approximation retaining* n_0 =2 *states, the offset relative to* p_c *satisfies*:

$$\eta_p = \frac{p_r - p_c}{p_c} < \frac{4f}{1 - 2\sqrt{2}f^2} \quad (16)$$

for all f < 1/2.

Comparing f with $E[N]$ (see subsection III.A), we find that the reciprocal of f equals transient failure rate (the average number of transient failures per peer). From equation (2), we know that $f > \mu / T$ since $\lambda > 0$. Further, to show the importance of failure distinction, we give a *rough* estimate of the worst recovery cost compared with that of oracle detector (refer to the appendix C to [13] for a detailed derivation).

Proposition 3: *Without failure distinction (e.g. timeout δ = 0), the extra cost relative to that of the oracle detector satisfies*:

$$\eta_c = \frac{c_{\max} - c_{\min}}{c_{\min}} \approx (1 + \frac{\mu}{fT}) \Big/ (1 - \frac{\mu}{fT}) \tag{17}$$

where c_{max} is the recovery cost without failure distinction, and c_{min} is the cost yielded by an oracle detector.

Based on the above propositions, we summarize some attractive features of our approximate solution:

Feature 1: *Our solution retains a minimal number of states when approximating the true replica availability, thus inducing the lowest computation cost.*

Feature 2: *Our solution provides a better tradeoff in systems with higher transient failure rates.* In particular, a smaller f imposes a stronger requirement on the quality of approximate solution (e.g., (17) indicates that $\eta_c \rightarrow \infty$ as $f \rightarrow \mu/T$). In our solution, $\eta_p \rightarrow 0$ as $f \rightarrow \mu/T$ given $T \gg \mu$ (indicated by (16)). This suggests the solution provides a better tradeoff to balance the increasing transient failure rate. Since P2P networks usually have high transient failure rate, our solution is well suited to address the cost-availability tradeoff in P2P systems.

Feature 3: *Our solution can sacrifice some quality to take the lowest complexity and computation cost road in stable systems with a low transient failure rate.* As transient failure rate $1/f$ decreases, the requirement on the quality of approximate solution is less stringent (e.g., as (17) indicates, given $T \gg \mu$, $\eta_c \approx 1$ as $1/f \rightarrow 0$). In such environment, our solution trades quality for the simplicity of solution.

F. Making Tuning Adaptive

The timeout estimation requires the knowledge of peer failure behavior, including p, μ and $F_c(x)$. In [13], we describe a method which can obtain these parameters directly from measurement trace, and this approach is periodically run to update these parameters. Accordingly, the system can adapt timeout value to the changing environment. An important remark is now in order. Deciding that a peer has left permanently within a finite trace essentially requires a threshold h for assuming that observing that a peer has left longer than h means that it has left permanently (we use h = 20 days in our evaluation to identify permanent failures in the first 10-day window). We would like to emphasize that h cannot be used as a timeout to navigate cost-availability tradeoff. This is because for a subsequent analysis of the historical trace, one would choose an extremely large h to identify permanent failures exactly. Online detection cannot afford such large thresholds, since it may put data availability in danger. Our evaluation result clearly shows this distinction.

G. Dealing with Repair Time

We present here a more accurate replication method for estimating target replication level t_r by considering the effect of repair time. Note that even a permanent failure can be immediately detected; the system still needs waiting a period to replace the failed peer. So the average replica availability over the period of peer lifetime and repair time decreases to Tp_c / $(T+R)$ where R is the average repair time. Clearly, to compensate for the reduced availability due to repair time, we must increase target replication level t_r:

$$t_r = \left\lceil \log(1-p_t) \Big/ \log\left(1 - \frac{R}{T+R} p_c\right) \right\rceil \tag{18}$$

In this paper, we mainly focus on reactive systems where repairs are fast (the reasons are given in [13]). Thus, repair time has little effect on the model given $R \ll T$. Besides, using a fine-gained t_r also helps to mask the effect of repair time on timeout selection. Simulations in Section IV further validate this intuition.

IV. EVALUATION

We evaluate the performance of our method through simulations based on both a failure model and actual system measurement traces. The two sets of simulations serve to validate different aspects of the advantages of our method. The simulations based on the failure model validate whether our method leads to a near-optimal tradeoff. The simulations based on actual system traces further validate whether our method still provides good performance when the system behavior does not exactly follow the failure model.

A. Evaluation using Failure Model

For simplicity, we first use a Markov failure model like other studies (e.g. [9, 20]), which assumes exponentially distributed session and downtime (Note that the SMP model described in Figure 1 can be applicable to general failure scenarios). With this model, we can plug in different set of transition rates to simulate different system environments.

We report three sets of simulations: (a) one for a P2P file-sharing environment such as Maze, with λ = 4.9 hours, μ = 14.1 hours, and T = 90 days; (b) one for a large corporation environment such as Microsoft Cooperation, with λ = 38 hours, μ = 13.5 hours, and T = 290 days; (c) and one for a wide-area collaboration environment such as PlanetLab, with λ = 8.5 days, μ = 3.5 days, and T = 213 days. Note that each setting has a very large transient failure rate $1/f$ (e.g. > 10 transient failures per peer).

Parameters of sets (b) and (c) are obtained from existing measurement [14, 16]. Parameters of set (a) are obtained from actual system traces [15]. Figure 3 shows the distribution of peer lifetime in the Maze system, which indicates that an exponential distribution is a reasonable fit for peer lifetime. Furthermore, for any lifetime distribution with a heavier tail, which includes Pareto, lognormal, Weibull, and Cauchy distributions, equation (14) can still be used. This formally follows from the fact that heavy-tailed lifetime imply stochastically larger residual lifetime $E[T_{2-}]$ in (11),

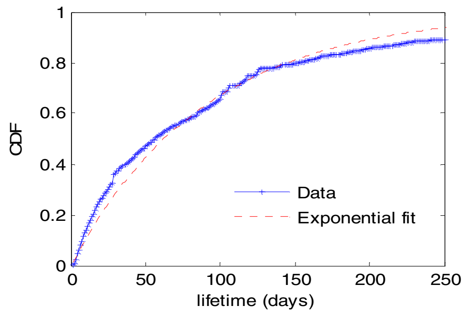

Figure 3. The CDF of peer lifetime.

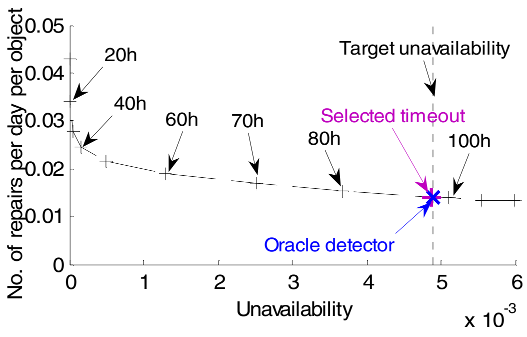

Figure 5. Recovery cost versus availability tradeoff with the synthetic trace for the Microsoft-like environment.

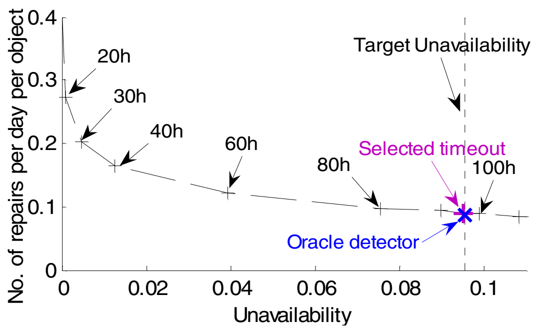

Figure 4. Recovery cost versus availability tradeoff with the synthetic trace for the Maze-like environment.

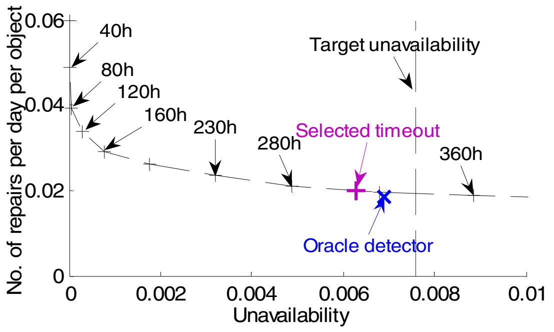

Figure 6. Recovery cost versus availability tradeoff with the synthetic trace for the PlanetLab-like environment.

which only makes our solution more conservative (e.g., a larger relative offset η_p) but still guarantees the target availability and reduces the recovery cost.

For each set, we use an exponential repair time with mean of one day and run Monte-Carlo simulations to evaluate performance of different timeout values. For the oracle detector, we assume that it knows exactly whether a failure is transient and permanent. We emphasize the performance of our estimated timeout (given by (14)) in each setting. Each run simulates 2000 data objects randomly scattered at start time across 1000 peers.

The initial number of replicas for each object is the target replica threshold t_r, which is computed according to (18). Note that t_r has very little effect since timeout is performed on a *per-replica* basis. We set two nines target availability for the settings (b) and (c) like the configuration in [6], but use a lower target (one nine) for the Maze system because it is more time-consuming to simulate such a highly dynamic system maintaining a higher level of data availability. Further, to eliminate the artificial boosting of the availability when taking the ceiling operation in (18), we revise the target availability p_t in each setting as follows: p_t = 0.9045 for the Maze system, p_t = 0.9951 for the Corporation system, and p_t = 0.9924 for the PlanetLab system. The particular values we choose for the target availability guarantee that when computing the target number of replicas t_r using (18), the value before taking the ceiling operation is already an integer (8 for the Maze system and 4 for the Corporation system and the PlanetLab system).

We run each simulation for a long simulated time period (100 days for the Maze-like environment and 300 days for the other two environments which have longer peer lifetime). Since permanent failures reduce the number of peers in the system, we also randomly add new peers with the same rate as permanent failures.

Figure 4 shows the first set of results for the Maze 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. Timeouts ranging from 1 hour to 120 hours are simulated. We see 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.

We see that our selected timeout (96 hours) provides a near-optimal tradeoff between availability and cost. Its availability is only slightly higher than the target (0.5% higher) and cost is low and extremely close to that of the oracle detector and the best timeout (3.3% higher than that of the oracle detector, and about 1.2% higher than that of the best timeout). To accurately identify a permanent failure in the historical trace, we use a very conservative threshold of 20 days. Clearly, this threshold is too large since any timeout larger than 100 hours fails to achieve the availability goal.

The results from Microsoft in Figure 5 and PlanetLab in Figure 6 show similar trends as the ones in Figure 4 (the solution in Figure 6 is relative more conservative due to a lower transient failure rate). Thus, our solution is suitable for different peer failure behaviors.

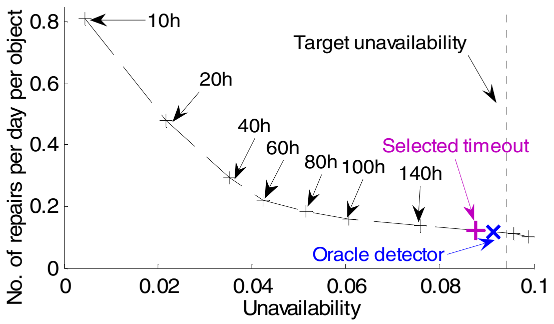

Figure 7. Recovery cost versus availability tradeoff by the real trace-driven simulation in the Maze environment

B. *Evaluation using System Traces*

In this section, we use actual system traces collected from Maze [15], which represents a typical highly dynamic P2P system since its dynamic level is close to Overnet [21]. We choose it as experiment environment because it has been monitored for a long period of time. We construct the environment with the system log from 3/1/2005 to 6/31/2005.

We still use 1000 peers in the system to store 2000 objects. To eliminate some extremely unstable peers (those that permanently leave after their first appearances), we only store objects on peers that have appeared at least two times. Peer failure characteristics are directly obtained from the historical behavior of system in March (refer to subsection III.D). The result is summarized in Figure 7.

Even the actual distribution is heavy-tailed [21], we see that the timeout (96 hours) given by equation (14) still provides a reasonable tradeoff between availability and cost: its recovery cost is 33% higher than the oracle detector. Therefore, for applications comfortable with such a cost increase, using such a timeout with closed-form expression is good enough.

To further provide a near-optimal tradeoff between availability and cost, we use the empirical downtime distribution, and solve (13) with the bisection method. The result shows that our new solution (164 hours) provides a slighter better availability than the target (1% higher), and a cost that is close to that of the oracle detector and the best timeout (4.3% higher than that of the oracle detector, and about 2.7% higher than that of the best timeout). To see to what extent our selected timeout saves cost, we also run the Carbonite algorithm [9], where the system does not distinguish failures but reuses the returning replicas. However, the cost of Carbonite is still 8.4 times greater than the oracle detector.

V. Related Work

There exists substantial work (e.g., [10-12]) on the study of failure detectors in networked systems, but they are quite different from detectors designed in the context of storage system, as stated in Section I. Several projects [4, 6, 8] try to mask transient failures by proactively adding a number of extra replicas. Chun et al. [9] suggest that reintegrating returning replicas can further reduce unnecessary replication. However, without the help of timeouts, these methods still incur high replication cost that is not necessary to maintain the desired level of availability.

Besides the timeout-based method presented in this paper, our prior work [22] proposed a probabilistic method which detects permanent failures. This method focuses mainly on the accuracy of *instantaneous* detections (e.g., to detect the *current* replication level of an object more accurately), but knows little about the impact of its false-positive or false-negative errors on *average* data availability and replication cost. In contrast, the timeout-based method in our present paper can directly navigate the cost-availability tradeoff on a long-term basis. More importantly, any probabilistic detection for an object has O(2^m) time complexity, where m is the total number of hosted replicas of the object. This high computational cost of the probabilistic method limits its practical applications to cases when each object has a large number of replicas. Given an availability goal, such cases can easily arise because many factors (such as high target replication level, inaccurate detections as well as memory-based repair policy) can lead to a large m, especially in highly dynamic P2P systems. In contrast, the timeout-based method performs each detection in O(m) time, resulting in a negligible computational overhead.

VI. Conclusion

In this paper, we show for the first time, to our knowledge, how to select timeouts to explore the cost-availability tradeoff. The advantage of our timeout selection method is that it can be performed on a per-replica basis. By pruning rare events, we can easily find a timeout which is capable of providing a near perfect balance between reduced availability due to detection delay and extra replication cost due to detection inaccuracy. Our results are very promising, showing that our selected timeout achieves close performance to that of the prefect oracle detector and the best timeout which strikes the perfect balance.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 60873051, the National Basic Research Program of China under Grant No.2004CB318204.

References

[1] Stribling J. OverCite: A Cooperative Digital Research Library. *Proc. of IPTPS*, 2005.

[2] Yan H, Tom Z. J. F, Dah-Ming C, John C. S. L and Cheng H. "Challenges, Design and Analysis of a Large-scale P2P-VoD system". *Proc. of SIGCOMM,* 2008.

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

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

[5] 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.

[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," *Proc. of OSDI,* 2002.

[8] 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.

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

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

[12] 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.

[13] Z. Yang, Y. Dai and Z. Xiao. Exploring the Cost-Availability Tradeoff in P2P Storage Systems. Technical Repaort TR-CS08-1100, Peking University, Nov. 2008 http://net.pku.edu.cn/2008.11/TR-CS08-1100.pdf

[14] Sit E, Haeberlen A, Dabek F, Chun B, Weatherspoon H, Morris R, Kaashoek M, and Kubiatowicz J. Proactive replication for data durability. *Proc. of IPTPS,* 2006.

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

[16] Bolosky W, Douceur J, Ely D, and Theimer M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. *Proc. of SIGMETRICS*, 2000.

[17] Bhagwan R, Savage S, and Voelker G. Understanding availability. *Proc. of IPTPS*, 2003.

[18] Ross.S. *Stochastic Processes*, 2nd ed. Wiley, 1996.

[19] EPC.Kao. *An Introduction to Stochastic Processes*. Wadsworth: Belmont, CA, 1997.

[20] Utard, G. and Vernois. A, Data durability in peer to peer storage systems. *Proc. of CCGrid.* 2004.

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

[22] J. Tian, Z. Yang, W. Chen, Y. Dai, B. Y. Zhao, "Probabilistic Failure Detection for Efficient Distributed Storage Maintenance " *Proc. of SRDS,* 2008.