Minimizing Churn in Distributed Systems

Philip Brighten Godfrey
Scott Shenker
Ion Stoica

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2005-12
http://www.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-12.html

November 12, 2005

Minimizing Churn In Distributed Systems

P. Brighten Godfrey, Scott Shenker, and Ion Stoica
{pbg,shenker,istoica}@cs.berkeley.edu

11th November 2005

1 Introduction

Peer-to-peer systems were designed from the outset to handle a high rate of churn: node joins, graceful leaves, and failures. Nevertheless, there is a price to churn which may manifest itself as dropped messages, increased user-experienced latency, or increased bandwidth use, potentially limiting the scenarios in which a P2P system is deployable [3]. Even in a reasonably stable managed infrastructure like PlanetLab [2], there can be a high rate of effective node failure due to nodes becoming extremely slow suddenly and unpredictably [9].

In this paper, we study how to reduce the churn rate itself. Specifically, we consider a scenario in which we desire to use k nodes, and we have $n \geq k$ available from which to choose. How should we select which k nodes to use in order to minimize churn among the chosen nodes over time? This question arises in many cases:

- Running a service on PlanetLab in which $n \approx 500$ nodes are available and we would like $k \geq 20$ to run the service in order to have sufficient capacity to serve requests.
- Selecting a reliable pool of $k \approx 1000$ super-peers from among $n \approx 100,000$ end-hosts participating in a peer-to-peer system.
- Connecting to one ($k = 1$) of n servers running an equivalent service.
- A node in an overlay network maintaining links to k of the n other nodes.
- Choosing k nodes to be nearest the root of an overlay multicast tree, where failures are most costly.
- In a storage system of n nodes, choosing k nodes on which to place replicas of a file.

In this paper we focus on churn as the main metric, leaving the effect of churn on other metrics of interest, such as latency, to future work.

We distinguish between two types of strategies for selecting the set of nodes to use. A *fixed strateg*y chooses a set of nodes and uses them for the entire run of the system (or a very long time). For example, in the PlanetLab scenario above, one might observe the nodes for some time prior to running the service, pick k that seem to be reasonably reliable, and then use that fixed set of nodes for months or years with little change. On the other hand, a *dynamic strategy* changes the set of nodes it uses over time. Consider what we call *Simple Random*: pick a random set of initial nodes; whenever a node we're using fails, replace it with a random available node. This dynamic strategy is particularly important for distributed systems because it requires no information about the history of nodes in the system.

We are by no means the first to select nodes to minimize the chance of failure. But most typically, systems simply employ the heuristic of picking the available node which has been up the longest [6, 5, 10], without a full exploration of the design space. We believe that there remain unaddressed questions, such as: (1) How much can churn be reduced by a dynamic strategy compared to a smart fixed one, and what is the ultimate effect on performance in a real system? (2) What information and selection algorithms are necessary to obtain a substantial benefit?

The answers could both explain the performance of current distributed system designs and provide guidance for future designs.

Results. It is hardly surprising that a sophisticated dynamic strategy can perform better than a fixed one. We quantify that improvement in real-world traces, showing a 2-6$\times$ reduction in churn over the best fixed strategy, and a corresponding reduction in the rate of failed route requests in a simulation of the Chord DHT [12].

Less obviously, even Simple Random can produce an average churn rate *arbitrarily better than the churn rate of the most reliable nodes.* It may in general also perform arbitrarily *worse* than fixed strategies when nodes have different mean lifetimes. Yet in real-world traces Simple Random consistently performs better than the best fixed strategy, in some cases by more than a factor of 3. In one of the two traces we study, its performance is close to that of the best dynamic strategies which we tested.

Additionally, we derive Simple Random's expected churn rate in a stochastic model (assuming a technical conjecture discussed in Section 3), and show that it closely matches our simulations.

The rest of this paper is as follows. In Section 2 we give our model of churn. In Section 3 we analyze fixed strategies and Simple Random. Section 4 contains our simulation results. We discuss related work in Section 5 and conclude in Section 6.

2 Model

System model. At any time, each of n nodes in the system is either *up* or *down*, and nodes that are up are either *in use* or *available*. Nodes fail and recover according to some unknown process. The current and past states of each node are known by the node selection algorithm. At any time, the selector may choose to add or remove a node from use, transitioning it from *available* to *in use* or back. There is a target number of nodes to be in use, $k = \alpha n$ for some $0 < \alpha \leq 1$, which the dynamic strategies we consider will match exactly unless there are fewer than k nodes up. The fixed strategies will pick some fixed set of k nodes, so they will have fewer than k in use whenever any picked node is down.

Performance metric. We define churn as follows. Given a sequence of changes in the set of in-use nodes, each change will consist of either node failures or reselections, but not both: a reselection in response to a failure happens subsequently as a separate event. Let U_i be the set of in-use nodes after the ith change, with U_0 the initial set. Then churn is the sum over each event of the *fraction of the system that has changed state* in that event, normalized by run time T:

$$C \quad = \quad \frac{1}{T} \cdot \sum_{\text{events } i} \frac{|U_{i-1} \ominus U_i|}{\max\{|U_{i-1}|, |U_i|\}},$$

where $\ominus$ is the symmetric set difference. So if each of k in-use nodes fail, one by one with no reselections, churn increases by $\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1} \approx \ln k$. If we then reselect k new nodes in one step, churn increases by $\frac{k-0}{k} = 1$.

Churn does not perfectly model every system, but it is a useful rule of thumb. For example, consider a storage system in which files are perfectly partitioned among nodes. Under some assumptions, churn is proportional to the probability that, when a random object is probed at a random time, the node (formerly) storing the primary replica of that object has recently failed or joined.

An important assumption is that a node which fails and then recovers is of no more use to us than a fresh node. This is reasonable for systems with state that is short-lived relative to the typical period of node downtime, such as in overlay multicast or $i3$ [11]. When applied to storage systems, a failure and recovery in our study should be taken to mean a permanent failure followed by the introduction of a new node. For the case that there is retention of storage across failures, see [4, 14].

3 Analysis

We analyze the best fixed strategy and Simple Random in the following renewal process. There is a distribution of lifetimes with given PDF f_i and mean μ_i for each node i. At time 0 all nodes are up. Each node draws a lifetime ℓ_1 from its distribution independently of all other nodes, fails at time ℓ_1, recovers instantaneously, draws another lifetime ℓ_2, fails at time $\ell_1 + \ell_2$ for an instant, and so on.

In this model, each failure and reselection costs $\frac{1}{\alpha n}$, so the best fixed strategy is simply to pick the k nodes with lowest μ_i. If all nodes have the same mean lifetime μ, all fixed strategies have churn $2/\mu$ (one failure and one reselection each μ units of time).

Two examples. On the other hand, Simple Random can do better: intuitively, rather than drawing a random lifetime from the underlying distribution, Simple Random selects the current lifetime of a random node. This is skewed towards longer lifetimes since a node spends longer in a long lifetime than in a short one. Consider, for example, a bimodal lifetime distribution which takes value $r \gg 1$ with probability $p = \frac{1}{r}$ and 1 otherwise. Any fixed strategy has churn $2/\mu \approx 1$. Now we'll sketch an analysis of Simple Random assuming $k = 1$ and large n. The fraction of time that a node spends in lifetimes of length r is $\frac{rp}{rp+1\cdot(1-p)} \approx \frac{1}{2}$, so when we pick a random replacement at a random time, we land in a lifetime of length r with probability $\approx \frac{1}{2}$, and one of length 1 otherwise. On average we'll land in the middle of the lifetime, so the effective mean lifetime is $\approx \frac{1}{2}(\frac{1}{2} \cdot r + \frac{1}{2} \cdot 1) \approx r/4$, resulting in churn $\approx 8/r$, which can of course be arbitrarily small.

But Simple Random does very badly when "good" nodes are rare. Suppose $k = 1$ and all nodes have exponential lifetimes, one with mean $r \gg 1$ and $n-1$ with mean 1. When Simple Random selects a node, the expected remaining lifetime is $\frac{1}{n}(r) + \frac{n-1}{n}(1) \approx 1$ when $n \gg r$, so its churn rate is 2. But the best fixed strategy has mean lifetime r and churn $2/r$.

Analysis of Simple Random. The following analysis assumes that the lifetime distributions have the property that the number of failures of in-use nodes within certain time periods concentrates around the mean, in the following sense.

Definition 1 *For a trial of length T, let C be the churn rate and L_i be a random lifetime of node i chosen uniformly at random over all lifetimes in the run. Let random variables X_i and R_i be the length of L_i and the number of reselections during L_i. Finally, let $c = \frac{\alpha n}{2} \cdot E[C]$ be the expected rate of reselections. Then the node lifetime distributions $f_1, \ldots, f_n$ are* good *if they have finite mean and variance, $E[C] > 0$, and $\forall i$,*

$$\Pr[(1-\varepsilon)cX_i \leq R_i \leq (1+\varepsilon)cX_i] \geq 1 - \varepsilon$$

$\forall \varepsilon > 0$, $\alpha \in (0, 1)$, and sufficiently large n and T.

This property is trivially true in the (uninteresting) case that all nodes have exponentially distributed lifetimes with common mean. We conjecture that in fact it is true quite generally. Our main analytical result is the following (proved in the Appendix).

Theorem 1 *Let C be the churn in a trial of length T using Simple Random. If the node lifetime distributions (f_i) are good and $\alpha \in (0, 1)$, then $E[C] =$*

$$(1 \pm \varepsilon)\frac{2}{\alpha n}\sum_{i=1}^{n}\frac{1}{\mu_i}\left(1 - \mathcal{L}[f_i(\ell)]\left(\frac{\alpha}{2(1-\alpha)}E[C]\right)\right),$$

$\forall \varepsilon > 0$ and sufficiently large n and T, where $\mathcal{L}[f(\ell)](x) := \int_{\ell=0}^{\infty} e^{-x\ell} f(\ell) d\ell$ is the Laplace transform.

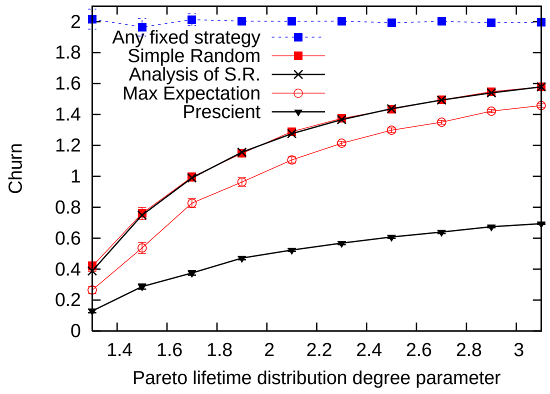

Figure 1: Simulation and analysis of churn with varying lifetime distribution, $n = 20$, and $\alpha = \frac{1}{2}$. Max Expectation and Prescient are defined in Section 4.1.

Figure 1 shows agreement of this analysis with a simulation for $n = 20$ and Pareto-distributed lifetimes with PDF $f(\ell) = ab^a/(x + b)^{a+1}$. We vary a and fix b so that $\mu = 1$. Even though the analysis assumes large n, it differs from the simulation by $\leq 1.5\%$ in nearly all cases, and is always within the simulation's 95% confidence intervals.

4 Simulations

We report simulation results using two metrics: (1) churn as defined in Section 2; and (2) the fraction of failed route operations in a simulation of the Chord DHT, to show performance in a real system.

We compare the node selection strategies defined in Section 4.1. Section 4.2 describes our simulation methodology. Our results appear in Section 4.3.

4.1 Selection strategies

Fixed strategies. In a situation such as deploying a service on PlanetLab, one could observe nodes for some time before running the system, and then use any of the following heuristics for selecting a "good" set of nodes to use for the lifetime of the system:

- *Fixed Random*: Pick k uniform-random nodes.
- *Fixed Decent*: Discard the 50% of nodes that were up least during the observation period. Pick k random remaining nodes.
- *Fixed Most Available*: Pick the k nodes that spent the most time up.
- *Fixed Longest Lived*: Pick the k nodes which had greatest average lifetime.

Dynamic strategies. The following strategies select a random initial set of k nodes, and pick a replacement only when an in-use node fails. They differ in which replacement they choose:

- *Simple Random*: Select a uniform-random available node.
- *Max Expectation:* Select the node with greatest expected remaining uptime, conditioned on its current uptime. Estimate this by examining the node's historical lifetimes.
- *Longest Uptime:* Select the node with longest current uptime. This is the same as Max Expectation when the underlying lifetime distribution has decreasing failure rate.

- *Prescient:* Select the node with longest time until next failure. This requires future knowledge, but provides a useful comparison. It is the optimal strategy when future knowledge is available (see Appendix).

4.2 Simulation setup

Due to space constraints full details appear in the Appendix.

Simulator. We tabulate churn in an event-based simulator which processes transitions in state (*down*, *available*, and *in use*) for each node. We also feed the sequence of events into a simple simulator of the Chord protocol included with the *i*3 [11] codebase. Once per simulated second we request that two random DHT nodes v_1, v_2 each route a message to the owner of a single random key k. The trial has *failed* unless both messages arrive at the same destination.

Failure patterns. For synthetic lifetime distributions we use the Pareto-distributed lifetimes of Figure 1, but with exponent $a = 1.5$ and mean 30 minutes. Between each lifetime we use exponentially-distributed downtimes with mean 2 minutes.

To evaluate the generality of our techniques, we also use traces from two very different environments. The PlanetLab All Pairs Ping trace [13] consists of pings sent every 15 minutes between all pairs of 200-400 PlanetLab nodes from January, 2004, to June, 2005. We consider a node to be up in one 15-minute interval when at least half of the pings sent to it in that interval succeeded. The second trace, from [1], is based on HTTP requests sent from a single machine at Carnegie Mellon to 129 major web sites every 10 minutes from September, 2001, to April, 2002.

In both traces, we split the trace in half, train the fixed strategies on the first half, simulate the strategies on the whole trace, and report statistics on the second half only.

4.3 Simulation results

Figure 2 shows results in the synthetic Pareto lifetimes as a function of α with fixed k, so that $n = k/\alpha$ varies. We can see that churn is roughly proportional to fraction of requests failed in Chord, for a fixed number of nodes k. In both Figure 1 and Figure 2, Simple Random is surprisingly close to Max Expectation when α is not small. As one would expect, performance is best when $\alpha \ll 1$. In this case, Max Expectation does much better than Simple Random intuitively because it finds the few nodes with very long remaining lifetime.

In the real-world traces, the parameter k does not directly control system size for the fixed strategies, since some nodes have extended downtimes. To provide a fairer comparison, we will plot performance as a function of the *average number of nodes in use over time*, controlled behind the scenes by varying k. Dynamic strategies have an advantage that this metric doesn't capture: the number of nodes in use is exactly k as long as there are $\geq k$ nodes up.

Figure 3 shows results for the PlanetLab trace. Simple Random has less than $\frac{1}{3}$ the churn of the best fixed strategy for a wide range of k, and there is a similar benefit in terms of failed requests in Chord, in addition to 3.3% lower mean message latency. (In Chord, more requests fail as k increases since route lengths increase.) Interestingly, there is little advantage in this trace to using more advanced dynamic strategies; with $k = 90$, Simple Random's churn is just 24% higher than the best non-prescient strategy, Longest Uptime.

Figure 4(a) shows churn in the web site trace. Here the fixed strategies which take into account per-node statistics perform significantly better than in PlanetLab. This is to be expected since, unlike the PlanetLab trace, the first half of the web site trace — on which the fixed strategies were trained — is a good predictor of performance in the second half. Figure 4(b,c) shows that this is not only because PlanetLab is growing.

A second difference is that Simple Random is noticeably worse than the other dynamic strategies. This is likely because there is more diversity among nodes: if we rank nodes by average lifetime, the 10th

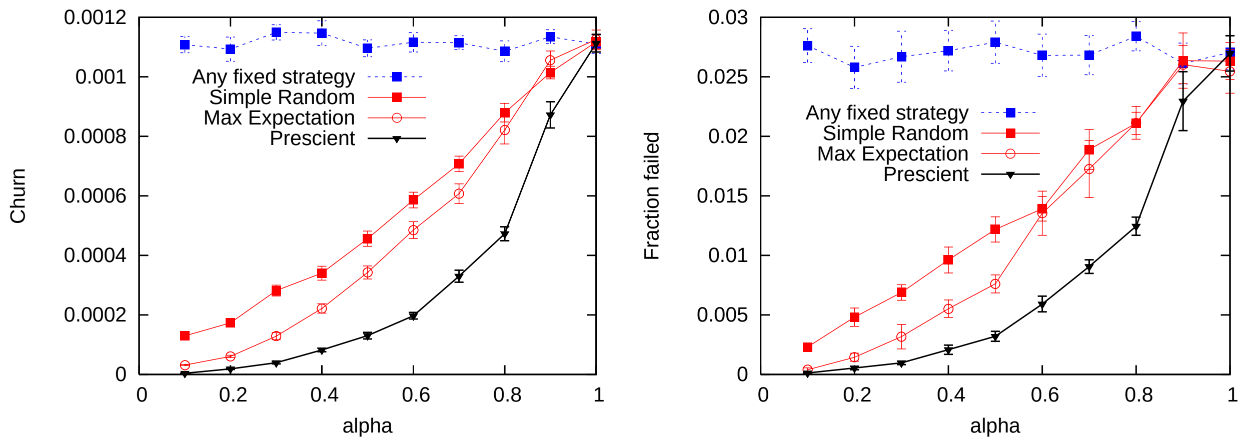

Figure 2: Churn (left) and fraction of requests failed in Chord (right) for varying α, with fixed $k = 50$ nodes in use and Pareto lifetimes.

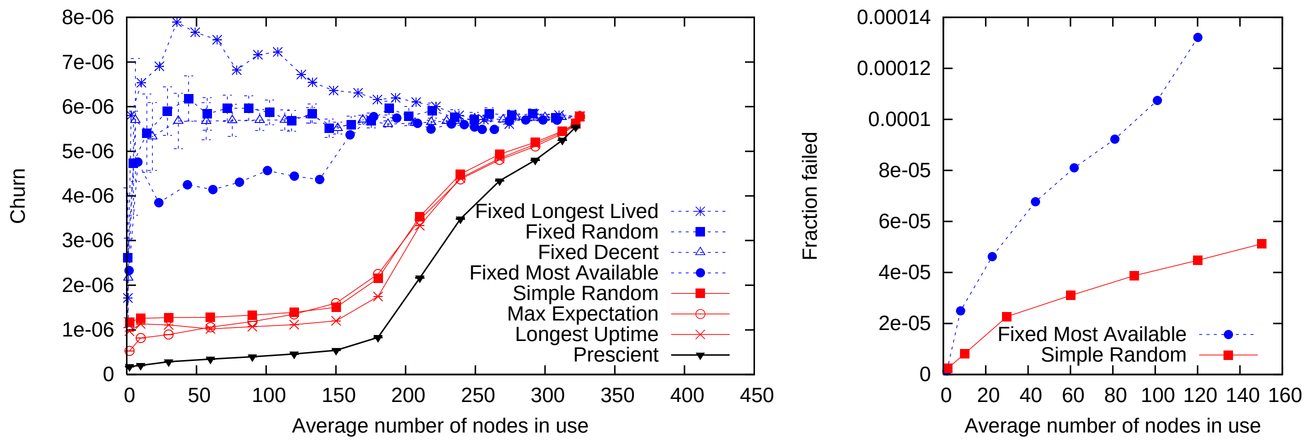

Figure 3: Churn (left) and fraction of requests failed in Chord (right) with varying average number of nodes in use in the PlanetLab trace. Chord results use a single trial per data point due to the length of time it takes to run the simulations.

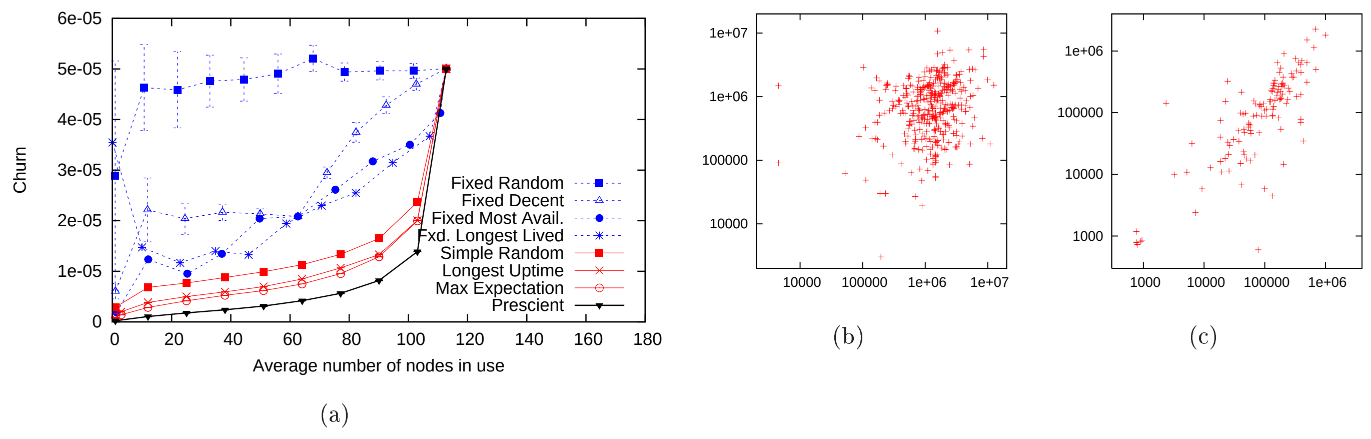

Figure 4: (a): Churn with varying average number of nodes in use in the web site trace. (b),(c): Scatterplots for the PlanetLab (left) and web site (right) traces: a point (x, y) is a node with average lifetime x sec in the first half and y sec in the second half of the trace. Not shown for PlanetLab are 91 nodes that were up only in the first half and 180 up only in the second half; the numbers are 2 and 2 for the web site trace.

and 90th percentile nodes in differ in average lifetime by a factor of 12.6 in PlanetLab and 35.8 in the web site trace. At the 5th and 95th percentiles, the factors are 27.3 and 541. Thus, there is more opportunity to utilize information about specific nodes. Nevertheless, once again Simple Random outperforms all of the fixed strategies.

5 Related work

Longest Uptime is a common heuristic which has been studied in context including of DHT neighbor selection [6], selecting superpeers [5], and selecting parents in an overlay multicast tree [10]. The Accordion DHT [7] selects neighbors by computing the conditional probability that a node is currently up given when it was last contacted and how long it was up before that, assuming lifetimes fit a Pareto distribution with learned parameters. Mickens [8] used complicated statistical techniques to predict future node uptime, and experimented with placing file replicas in Chord on successors with greatest predicted time to live.

Weatherspoon et al [14] share our goal of minimizing churn, but in the context of storage systems. They avoid much of the unnecessary cost of transient failures (where a node fails and recovers with data intact) by delaying rereplication actions. In contrast, our results are not specific to storage systems, but we model only non-transient failures.

6 Conclusion

We have highlighted the distinction between fixed and dynamic node selection strategies and shown that the latter can offer a significant reduction in churn. Moreover, we have demonstrated the surprisingly good performance in theory and in practice of the Simple Random strategy, which uses no information about nodes' history. One avenue of future work is to compare fixed and random selection strategies in a real deployment. It would also be interesting to apply our study to the other application scenarios listed in Section 1.

We thank Anwitaman Datta and Hakim Weatherspoon for helpful discussions, and the authors of [1, 13] for supplying their traces.

References

[1] M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, May 2002.

[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Proc. NSDI*, March 2004.

[3] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer neetworks: Pick two. In *Proc. HOTOS*, May 2003.

[4] C. Blake and R. Rodrigues. High availability in DHTs: Erasure coding vs. replication. In *Proc. IPTPS*, 2005.

[5] L. Garces-Erice, E. W. Biersack, K. W. Ross, P. A. Felber, and G. Urvoy-Keller. Hierarchical P2P systems. In *Proc. ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, Klagenfurt, Austria, 2003.

[6] J. Ledlie, J. Shneidman, M. Amis, M. Mitzenmacher, and M. Seltzer. Reliability- and capacity-based selection in distributed hash tables. Technical report, Harvard University Computer Science, 2003.

[7] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek. Bandwidth-efficient management of DHT routing tables. In *Proc. NSDI*, 2005.

[8] J. Mickens and B. Noble. Predicting node availability in peer-to-peer networks. In *ACM SIGMETRICS poster*, 2005.

[9] S. Rhea, B.-G. Chun, J. Kubiatowicz, and S. Shenker. Fixing the embarrassing slowness of OpenDHT on PlanetLab. In *Proc. WORLDS*, 2005.

[10] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proc. SIGCOMM*, 2004.

[11] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. SIGCOMM*, 2002.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, 2001.

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

[14] H. Weatherspoon, B.-G. Chun, C. W. So, and J. Kubiatowicz. Long-Term Data Maintenance: A Quantitative Approach. Technical Report UCB/CSD-05-1404, Computer Science Division, UC Berkeley, July 2005.

Appendix A: Trace processing details

Simulator. We tabulate churn in an event-based simulator which processes transitions in state (between *down*, *available*, and *in use*) for each node.

We also feed the sequence of events into a simple simulator of the Chord protocol, a slight modification of the one included with the *i*3 [11] codebase. Events are node joins and failures and datagrams being sent and received. Datagram delivery is exponentially distributed with mean 50 ms between all node pairs with no loss (unless the recipient fails while the datagram is in flight). We request at a rate of once per simulated second that two random DHT nodes v_1, v_2 each route a message to the owner of a single random key k. We consider the result *successful* if both messages arrive at the same destination.

Failure patterns. For synthetic lifetime distributions we use the Pareto-distributed lifetimes of Figure 1, but with exponent $a = 1.5$ and mean 30 minutes. Between each lifetime we use exponentially-distributed downtimes with mean 2 minutes.

We also two traces. The PlanetLab All Pairs Ping trace [13] consists of pings sent every 15 minutes between all pairs of 200-400 PlanetLab nodes. We use the portion of the trace between January, 2004, and June, 2005. We consider a node to be up in one 15-minute interval when at least half of the pings sent to it in that interval succeeded (and at least one ping was attempted). In a number of periods, all or nearly all PlanetLab nodes were down, most likely due to planned system upgrades or measurement errors. To exclude these cases, we "cleaned" the trace as follows: for each period of downtime at a particular node, we remove that period (i.e. we consider the node up during that interval) when the average number of nodes up during that period is less than half the average number of nodes up over all time.

The second trace, from [1], is based on HTTP requests sent from a single machine at Carnegie Mellon to 129 major web sites every 10 minutes from September, 2001, to April, 2002. Since there is only a single source, network connectivity problems near the source result in periods when nearly all nodes are unreachable. We attempt to remove such measurement problems using the same heuristic with which we cleaned the PlanetLab data.

In both traces, we split the trace in half, and allow the fixed strategies to examine the first half in order to decide which nodes to use. We then simulate the strategies on the whole trace, reporting statistics on the second half only.

Plots. Unless otherwise specified, plots use at least 10 trials and show 95% confidence intervals.

Appendix B: Proofs

Proof of Theorem 1

We will use the fact that in our stochastic model, each failure and reselection costs $\frac{1}{\alpha n}$, so counting both types of events and averaging over time, the churn rate is

$$C = \frac{2}{T\alpha n} \cdot (\text{total \# of failures of in-use nodes}). \tag{1}$$

Proof: (of Theorem 1) We will analyze the expected churn using the fact that the number of failures is equal to the total number of times a node is selected. Let L_i be a lifetime of node i selected uniformly at random over all its lifetimes in the run of the system. (We will abuse notation and refer to L_i as both a lifetime and the length of that lifetime.) Let R_i be the number of reselections during L_i, and let $\beta = 1-\alpha$. Finally, define

$$S_i \;=\; \begin{cases} 1 & \text{if } i \text{ is selected in lifetime } L_i \\ 0 & \text{o.w.} \end{cases}$$

Since each reselect picks one out of the βn available nodes u.a.r., if there are r selections during a particular node's lifetime, the probability it is selected during that lifetime is

$$\begin{aligned} \Pr[\text{node } i \text{ picked after } r \text{ uniform-random reselects}] &= 1 - \Pr[\text{none of } r \text{ reselects picks node } i] \\ &= 1 - \left(1 - \frac{1}{\beta n}\right)^r, \\ &\leq (1+\varepsilon)\left(1 - e^{-r/\beta n}\right), \end{aligned} \tag{2}$$

for any $\varepsilon > 0$ and sufficiently large n (since $\beta > 0$). Thus,

$$\begin{aligned} E[S_i] &= E\left[S_i \cdot 1_{(R_i \leq (1+\varepsilon)cL_i)} + S_i \cdot 1_{(R_i > (1+\varepsilon)cL_i)}\right] \\ &\leq E\left[(1+\varepsilon)\left(1 - e^{-(1+\varepsilon)cL_i/\beta n}\right)\right] + \Pr[R_i > (1+\varepsilon)cL_i] \quad \text{(Eq. 2)} \\ &\leq (1+\varepsilon)^2 E\left[1 - e^{-cL_i/\beta n}\right] + \varepsilon \qquad \text{(since } f_i\text{'s are good)} \\ &\leq (1+\varepsilon) E\left[1 - e^{-cL_i/\beta n}\right] + \varepsilon, \end{aligned} \tag{3}$$

where in the last step we have reset ε appropriately for notational convenience (we will continue such implicit merging below). Now letting N_i be the number of lifetimes that node i has in time $[0, T]$, by Equation 1, we have

$$\begin{aligned} E[C] &= \frac{2}{\alpha n T} \sum_{i=1}^{n} E[N_i S_i] \\ &= \frac{2}{\alpha n T} \sum_{i=1}^{n} E\left[N_i S_i \cdot 1_{(N_i \leq (1+\varepsilon)EN_i)} + N_i S_i \cdot 1_{(N_i > (1+\varepsilon)EN_i)}\right] \\ &\leq \frac{2}{\alpha n T} \sum_{i=1}^{n} \left((1+\varepsilon)(EN_i)E[S_i] + E\left[N_i \cdot 1_{(N_i > (1+\varepsilon)EN_i)}\right]\right) \\ &\leq \frac{2}{\alpha n T} \sum_{i=1}^{n} \left((1+\varepsilon)(EN_i)E[S_i] + \varepsilon E[N_i]\right), \end{aligned}$$

where the last step follows from the Central Limit Theorem, which we can apply since we have assumed finite mean and variance of the distribution f_i. Now by the Strong Law of Large Numbers, $EN_i/T \to 1/\mu_i$ as $T \to \infty$, so

$$\begin{aligned} E[C] &\leq \frac{2}{\alpha n T} \sum_{i=1}^{n} \frac{(1+\varepsilon)T}{\mu_i} (\varepsilon + E[S_i]) \\ &\leq \frac{2}{\alpha n} \sum_{i=1}^{n} \frac{(1+\varepsilon)}{\mu_i} \left(\varepsilon + (1+\varepsilon)E\left[1 - e^{-cL_i/\beta n}\right] + \varepsilon\right) \quad \text{(by Eq. 3)} \\ &\leq O(\varepsilon) + (1+\varepsilon')\frac{2}{\alpha n} \sum_{i=1}^{n} \frac{1}{\mu_i} \left(\int_0^\infty 1 - e^{-c\ell/\beta n} f_i(\ell)\, d\ell\right), \end{aligned}$$

for any $\varepsilon' > 0$ and sufficiently small ε. Since $E[C] > 0$ by assumption, we can choose ε sufficiently small and absorb the additive $O(\epsilon)$ term into the $(1+\varepsilon')$ factor. A similar technique gives the lower bound

$$E[C] \quad \geq \quad (1-\varepsilon)\frac{2}{\alpha n}\sum_{i=1}^{n}\frac{1}{\mu_i}\int_0^{\infty}\left(1-e^{-c\ell/\beta n}\right)f_i(\ell)d\ell.$$

The result follows after substituting $c = \frac{\alpha n}{2} \cdot E[C]$ as given in Definition 1. ∎

Facts concerning dynamic strategies

We require a dynamic strategy to keep k nodes in use whenever $\geq k$ are up, and as many as possible otherwise. So during any time period in which $\leq k$ nodes are up, all (legal) dynamic strategies have the exact same behavior and the same churn. For the Facts that follow, we may therefore restrict our attention to the case that there are always $\geq k$ nodes up. In this case, each failure and reselect costs $1/k$, and we can think about the strategy of in-use nodes as k *chains* $v^1, \ldots, v^k$ where each $v^i = (v^i_1, \ldots, v^i_{m_i})$ is such that v^i_j is selected in response to the failure of v^i_{j-1}.

A (graceful) *leave* is an event in which the selection algorithm decides to transition a node from *in use* to *available.* By the definition of churn in Section 2, a failure costs as much as a leave, which results in the following.

Fact 1 *Fix the pattern of failures. Suppose some strategy of node selections (v^i_j) leaves a machine and has churn C. Then there is another strategy (w^i_j) which never leaves and has churn $\leq C$.*

Proof: Let v^i_j be a node which is left at some time t_1 before its next failure at time t_2. Consider two cases: (1) node v^i_j is not used during $[t_1, t_2)$. Then let v^i_ℓ, $\ell > j$, be the node in use in chain i at time t_2 by strategy v. Form strategy w by deleting all nodes in chain i between j and ℓ, and just staying on node v^i_j during $[t_!, t_2)$. The two strategies differ only during $[t_1, t_2)$, during which v incurs at least the cost of one leave, while w incurs at most the cost of one failure. Case (2): node v^i_j is used again during $[t_1, t_2)$. If it is used again by chain i, we need only delete any intervening nodes in the chain to form w. Otherwise, if it is used by some other chain ℓ, we can swap the chains as follows: chain i stays on node v^i_j until it fails, and then continues following chain ℓ's selections from time t_2 onward. Chain ℓ, rather than switching onto node v^i_j, follows chain i's former selections. Clearly constructing strategy w in this way can only reduce the total number of failures and reselections. Iterating this argument completes the proof. ∎

Fact 2 *With full knowledge of the future, the Prescient strategy of Section 4.1 is optimal.*

Proof: Let v^i_j be a node which is selected at time t_1 and fails at some future time $t_2 > t_1$, and at the time it is selected, there is an available node u which next fails at time $t_3 > t_2$. If u is never in use during $[t_1, t_3)$ then clearly we can use u instead of v^i_j, and resume following chain v beginning at time t_3. Otherwise, suppose u is used during $[t'_1, t_3)$ for some $t'_1 > t_1$ by some chain v^ℓ. Then we can modify chain v^i to use u from t_1 until it fails at time t_3, and thereafter follow the former chain v^ℓ; and we can modify chain v^ℓ to follow the former chain v^i beginning at time t'_1. This does not introduce any new failures. Iterating the argument completes the proof. ∎

Facts concerning fixed strategies

Definition 2 *The decision problem* Best Fixed Strategy *(BFS) is as follows:*

- ***Instance:*** *A set V of n nodes; for each node $v \in V$ a sequence of failure and recovery times $f_1 < r_1 < f_2 < r_2 \ldots$; an integer k; and a rational c.*

- ***Question:*** *Does there exist a set $S \subseteq V$ of $\geq k$ nodes such that, when using S under the given pattern of failures, the churn incurred is $\leq c$?*

One might object that this definition is not realistic, since k does not directly control the number of nodes in use: for example, the definition allows picking a set of k nodes that are always down. But the proofs that follow do not make use of such pathological cases, and transfer directly to the variant of the problem where $|S|$ is unconstrained but we are required to have an average of $\geq k$ nodes in use over time.

Fact 3 BEST FIXED STRATEGY *is **NP**-complete.*

Proof: Clearly the problem is in **NP**. To show **NP**-hardness, we reduce from MAX CLIQUE, an instance of which consists of a graph $G = (V, E)$ and a clique size s. First assume that *exactly* k nodes are required by BFS. We reduce the MAX CLIQUE instance to an instance of BFS as follows:

Set $k = s$. There are $n = |V|$ nodes in the BFS instance identified with the n nodes in G. All nodes are up all the time, except as we will specify. For each pair of nodes $v, w \in V$, we set aside a period of time P_{vw} in the trace during which each node other than v and w fails and recovers, all at independent times. Let $[t_1, t_4]$ be some arbitrary subinterval of P_{vw} during which there are no failures. Then if $(v, w) \in E$, we have v and w fail and recover at independent times during $[t_1, t_4]$. Otherwise, they are down during overlapping periods, according to the following sequence of events, where $t_1 < t_2 < t_3 < t_4$:

- t_1: v fails;
- t_2: w fails;
- t_3: v recovers;
- t_4: w recovers.

Now suppose we pick some set $S \subseteq V$ of k nodes to use. Note that if both v and w are in S and $(v, w) \in E$, or if one of v or w is not in S, then the churn during P_{vw} is $n \cdot 2/k$ since each node fails and recovers at independent times (so each event costs $1/k$). However, if $v, w \in S$ and $(v, w) \notin E$, then v and w have overlapping failures and the churn is $(n-1) \cdot \frac{2}{k} + \frac{2}{k-1} > 2n/k$. Summing over all $n(n-1)$ periods, we have that if S corresponds to a k-clique in G, then the churn is $n(n-1)n\frac{2}{k}$, but otherwise the churn is strictly greater. Thus, asking for a k-clique in G is equivalent to asking whether there exists a fixed set of k nodes such that the churn incurred when using S is $\leq n(n-1)n\frac{2}{k}$.

To handle the case that $> k$ nodes are permissible, we can construct additional failures such that the number of nodes chosen will dominate the churn, forcing $|S| = k$. To do this, in a "fresh" time period with no other failures, we have all the nodes fail sequentially, and then recover sequentially in the reverse order. Regardless of which nodes are chosen, the same pattern arises among the chosen nodes, for a churn during this period of $\Theta(\log |S|)$. Repeating this pattern $\Theta(n(n-1)n\frac{2}{k})$ times is sufficient to ensure that regardless of the pattern of failures representing the graph structure, a smaller S has lower churn, so the optimal S has $|S| = k$. ■

Fact 4 *Picking the k nodes with fewest failure and recovery events is a k-approximation for* BEST FIXED STRATEGY.

Proof: Each event costs ≤ 1, while the optimal strategy must pick a set of nodes with at least as many events, each with cost $\geq \frac{1}{k}$. ■