Replication Degree Customization for High Availability*

Ming Zhong†
Google, Inc.
mzhong@google.com

Kai Shen
Dept. of Computer Science
University of Rochester
kshen@cs.rochester.edu

Joel Seiferas
Dept. of Computer Science
University of Rochester
joel@cs.rochester.edu

ABSTRACT

Object replication is a common approach to enhance the availability of distributed data-intensive services and storage systems. Many such systems are known to have highly skewed object request probability distributions. In this paper, we propose an object replication degree customization scheme that maximizes the expected service availability under given object request probabilities, object sizes, and space constraints (*e.g.*, memory/storage capacities). In particular, we discover that the optimal replication degree of an object should be linear in the logarithm of its popularity-to-size ratio. We also study the feasibility and effectiveness of our proposed scheme using applications driven by real-life system object request traces and machine failure traces. When the data object popularity distribution is known a priori, our proposed customization can achieve 1.32–2.92 "nines" increase in system availability (or 21–74% space savings at the same availability level) compared to uniform replication. Results also suggest that our scheme requires a moderate amount of replica creation/removal overhead (weekly changes involve no more than 0.24% objects and no more than 0.11% of total data size) under realistic object request popularity changes.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability.

General Terms

Experimentation, performance, reliability.

*This work was supported in part by the U.S. National Science Foundation (NSF) grants CCR-0306473, ITR/IIS-0312925, CAREER Award CCF-0448413, CNS-0615045, CCF-0621472, and by an IBM Faculty Award.

†This work was done while the author was at the University of Rochester.

EuroSys'08, April 1–4, 2008, Glasgow, Scotland, UK.

Keywords

System availability, replication, optimization.

1. INTRODUCTION

Object replication is commonly employed to enhance the availability of data-intensive services, on wide-area networks [2, 7, 16, 18, 21, 36], on mobile and wireless networks [17], on enterprise-area networks with decentralized management [3, 10], and on centrally-managed local-area clusters [12, 13, 26, 29–32]. However, existing availability-oriented replication schemes are often oblivious to object request popularities when determining object replication degrees. Typically, all objects are assigned the same number of replicas. For example, CFS [7] uniformly maintains k ($k = 6$ in their settings) replicas for each file block. Farsite [3,10], guided by its measurement results on desktop free spaces, uses 3 or 4 replicas per file.

However, many large-scale data-intensive applications contain objects with highly skewed data popularity distributions. Such popularity skewness may be exploited to improve system availability under a given space constraint. Intuitively, using more replicas for popular objects (less for unpopular ones) can increase the overall expected service availability[1] while keeping the total space cost unchanged. The space constraint here may concern the external storage space or memory space when the dataset must reside in memory for fast response (as in some interactive Internet services).

Under space constraint, the popularity-adaptive object replication degree customization must also consider object sizes, particularly when object popularity and size distributions are correlated. Using a simple model, we propose a replication degree policy that maximizes the availability for systems with known object popularities and sizes. Specifically, we find that the optimal number of replicas for each object i should be linear in $\log \frac{r_i}{s_i}$, where r_i is the normalized object popularity and s_i is the normalized object size. We further consider our result in the context of additional practical application semantics (multi-object service requests) and heterogeneous systems (nonuniform/correlated machine failures).

[1]In some systems like Farsite [10], the availability metric is defined as the availability of the least available object in the whole system. Here we use a service-centric availability metric called *expected service availability* — the proportion of all successful service requests (each of which accesses one or more data objects) over all requests.

Object popularities in a system may change over time and such changes may call for adjustments on object replication degrees. In order for popularity-adaptive object replication degree customization to be practical, the adjustment overhead associated with object popularity changes must be limited. Additionally, a popularity-adaptive replication scheme tends to select higher replication degrees for more popular objects. This adversely affects write requests by increasing the replication consistency maintenance overhead. In this paper, we derive variants of our proposed replication scheme to address these feasibility concerns.

Our contribution is two-fold. First, we propose a novel popularity/size-adaptive object replication degree customization scheme for high service availability. Second, we evaluate its feasibility and effectiveness under practical system environments. Our quantitative evaluation uses applications driven by large real-life system data object request traces (IRCache web proxy cache [15] request trace, Ask.com [1] web keyword search trace, Harvard NFS file access trace [11], and CMU Coda file system [17] access trace). The node failure models in our evaluation are driven by failure/inaccessibility traces of real distributed systems (Ask.com server machines and Planetlab machines) and a synthetic WAN service failure model [8].

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 provides background and motivation for popularity/size-adaptive object replication degree customization. Section 4 derives an optimal replication degree policy using a simplistic model and then we further consider our result in practical application and distributed system contexts. Section 5 addresses additional practical concerns beyond achieving high availability. Section 6 quantitatively assesses the benefit and overhead of the proposed replication scheme. Section 7 concludes the paper.

2. RELATED WORK

There is a large body of previous work concerning object replication for high availability in distributed systems. The Farsite project [3, 10], Mickens and Noble [20], and Bhagwan *et al.* [2] all studied the estimation of machine availability in a distributed system and its utilization in guiding object replication degree and replica placement. Yu and Vahdat [37] investigated the relationship between availability and a number of factors including replication maintenance overhead. Nath *et al.* [21] measured correlated machine failures in real systems and studied its implication on data replication for high availability. Overall, existing availability-oriented replication systems or techniques do not consider object request popularities when determining per-object replication degrees. In particular, many systems (*e.g.*, Farsite [3], CFS [7], GFS [12], Glacier [14], IrisStore [21], MOAT [36]) use a fixed number of replicas for every object.

Object replication also serves other purposes than enhancing availability. For instance, Glacier [14] employs object replication for improving storage durability. Cohen *et al.* [6] proposed a square-root principle in determining customized object replication degrees for short search latency in peer-to-peer networks. The replication strategy in Beehive [24] exploits power law query distributions to achieve constant object look-up time in distributed hash tables. Kangasharju *et al.* [16] utilized the object popularities in determining replication strategies for small object access cost. Generally speaking, different system models and objective metrics demand studies on very different sets of issues. In particular, prior studies on object replication degrees have reached different results from ours since their optimization goals differ from ours — maximizing availability under space constraint.

Replica consistency has also been heavily studied in the past. Some have employed relaxed replica consistency semantics [12,31,32] to achieve high service throughput. Others seek to achieve high throughput and strong replica consistency at the same time (through application-specific data replication [13,29] or using chain replication [26]). Determining object replication degrees for high availability is mostly an orthogonal issue to replica consistency, except that a higher replication degree requires more consistency maintenance overhead for write-intensive services.

As an alternative to full object replication for data redundancy, erasure coding [22, 23] represents an object by n fragments, out of which any m fragments can be used to reconstruct the object (typically $1 < m < n$). Weatherspoon and Kubiatowicz [35] suggested that systems employing erasure coding can have much higher availability than replicated systems with similar storage requirements. However, a later study by Rodrigues and Liskov [27] showed that erasure coding is not always preferable. In particular, erasure coding requires more network bandwidth and computation cost; it introduces additional service delay; and it is inappropriate when partial object retrieval is desirable (*e.g.*, retrieving a few top matches in an index file for a search query). In this paper, we focus on improving availability through full object replication.

3. BACKGROUND AND MOTIVATION

3.1 Targeted Systems

Generally speaking, our adaptive object replication study is applicable to any data-intensive distributed systems with potential machine failures and with space constraints for data storage (so one cannot replicate all objects with an arbitrarily high degree to achieve desired availability). Here we describe two common distributed system environments that are specifically targeted in this paper. We discuss their machine failure properties and explain their space constraints.

- *Decentralized wide-area/enterprise-area systems.* Due to uncoordinated shutdown/maintenance and network connectivity problems, the per-node failure rate in wide-area distributed systems can be quite high (averaging at 0.265 from a four-month Planetlab machine accessibility trace, described later in Section 6.1). Non-uniform machine failures are commonplace since nodes in these systems can have very different physical configurations and network accessibility. Additionally, correlated machine failures are also possible due to common failure causes like workload-triggered software bug manifestations and Denial-of-Service attacks [21]. As for the space constraint, the high per-node failure rate in these systems may demand high-degree replication to achieve desired service availability. Without proper optimization, this may result in high storage demand and excessive cost of placing/maintaining the replicas.
- *Centrally-managed local-area clusters.* Many data-intensive services run on centrally-managed local-area clusters. Examples include web keyword search engines and digital libraries. With central management, the per-node

Traces	Num. of distinct objects	Num. of requests	Collection time periods
IRCache web proxy [15]	4.89 million web pages	33.46 million	09/15/2005–10/27/2005 (six weeks)
Ask.com Search [1]	1.41 million keyword indexes	75.92 million	01/01/2006–02/25/2006 (eight weeks)
Harvard NFS [11]	667,656 files	2.78 million	02/01/2003–03/14/2003 (six weeks)
Coda distributed FS [17]	16,250 files	556,703	01/01/1993–01/31/1993 (four-five weeks)

Table 1: Real system data object request traces and their statistics.

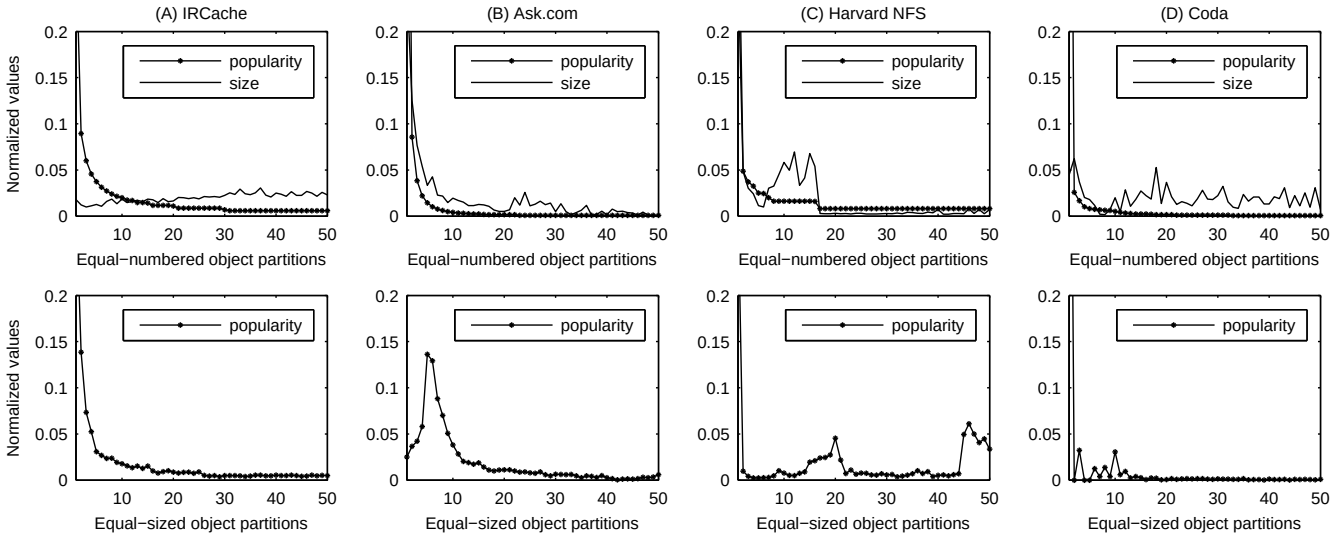

Figure 1: An illustration on object request popularity skewness. For each trace, the objects are first sorted in the descending order of their popularities and then partitioned into 50 equal-numbered (the upper row) or equal-sized (the lower row) groups. The absolute values of popularity and size for each object partition are both normalized to those of all object partitions combined.

failure rate in these systems is relatively low (averaging at 0.000455 from a six-month Ask.com server failure trace, described later in Section 6.1). Non-uniform machine failures are not as likely in these systems but correlated failures may still exist. As for the space constraint, these systems typically do not require very high-degree replication to achieve desired service availability. However, it is a common requirement to keep dataset in memory to support fast response for interactive Internet users. Therefore the space constraint here concerns memory space, which is far more precious than external storage space.

In addition to the system environments, application-level service semantics may also affect our study. In particular, a service request may involve multiple data objects and typically the request succeeds only if all objects are available. An example is the multi-keyword search query (in a web search engine) that requires the availability of data indexes associated with all keywords in the query. Multi-object requests have subtle implications on system availability [36] and we must consider them in our object replication scheme.

3.2 Primary Motivations — Object Popularity Skewness and Stability

Many real-world data-intensive applications exhibit high skewness in their object request popularities. To illustrate this quantitatively, we collect data object request traces for four such systems (IRCache web proxy cache [15] request trace, Ask.com [1] web keyword search trace, Harvard NFS file access trace [11], and CMU Coda file system [17] access trace). Table 1 lists some statistics of these traces. The object request popularity skewness is illustrated in Figure 1. The upper row shows that the top 10% most popular objects account for at least 92% system requests in all traces. Due to such skewness, the overall expected service availability may be increased by popularity-adaptive replication — higher replication degrees for more popular objects.

Under space constraint, object sizes also need to be considered when object popularity and size distributions are correlated. Such correlation is evidenced by the difference between size-normalized object popularities (the lower row of Figure 1) and original object popularities (the upper row). This correlation is strongest for the Ask.com trace in Figure 1(B), where a popular keyword tends to be associated with a large data index (list of web pages containing the keyword). In such cases, it may not be preferable to use high replication degrees for popular objects since the large space consumption diminishes or even outweighs the availability increase.

The popularity of individual data objects may not stay constant over the lifetime of a data-intensive service. For instance, some objects may only become popular for short bursts of time (*e.g.*, news pieces and security patches for short-lived viruses). For writable services, new data ob-

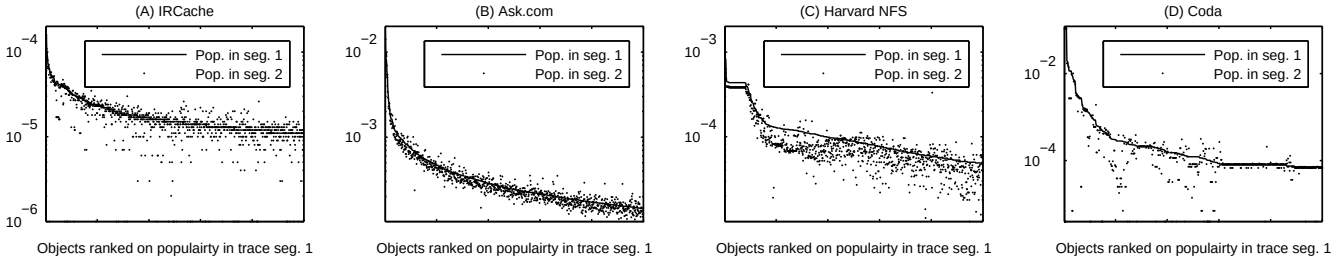

Figure 2: An illustration on object popularity stability. For each trace, we divide it into two equal-length segments. For the most popular 1000 objects (500 for Coda) in trace segment 1, we compare their popularities in the two trace segments. The closeness between the dots (popularity in trace segment 2) and the curve (popularity in trace segment 1) indicates small popularity changes.

jects that did not exist before may enter the system and become popular. For a popularity-adaptive object replication scheme, the popularity change of individual data objects may call for replication degree adjustment and thus incur replica creation/deletion overhead. In addition to the replica adjustment overhead, unstable object popularities also make it difficult to estimate such popularity values that are used to determine the object replication degrees.

We analyze our object request traces to assess the object popularity stability in real systems. The four traces listed in Table 1 are over four to eight weeks long. For each trace, we divide it into two equal-length segments and we then examine the object popularity changes between the two trace segments. Figure 2 illustrates such changes for the most popular objects. Quantitatively, we define the variation metric between two distributions π_1 and π_2 as:

$$||\pi_1, \pi_2|| = \frac{\sum_i |\pi_1(i) - \pi_2(i)|}{\sum_i \pi_1(i) + \sum_i \pi_2(i)} \quad (1)$$

Note that $||\pi_1, \pi_2|| \in [0, 1]$. The variation between the two trace segments is 0.124, 0.068, 0.167, and 0.108 for IRCache, Ask.com, Harvard NFS, and Coda respectively.

Our trace analysis suggests that the popularities for most data objects tend to remain largely stable over multi-week periods but object popularity changes do exist and their associated costs must be considered for popularity-adaptive object replication.

4. OBJECT REPLICATION DEGREE CUSTOMIZATION

We present an object replication degree customization scheme that achieves high expected service availability under given space constraint, object request popularities, and object sizes. We first provide an optimal result using a simple model — each service request accesses a single object; machines fail independently with a uniform failure probability. We then further consider this result in the context of practical issues including multi-object requests, nonuniform machine failure rates, and correlated machine failures.

4.1 A Simple Model — Single-Object Requests; Uniform Independent Machine Failures

Our system model assumes independent machine failures with a uniform failure probability of $0 < p < 1$ — each machine fails to answer requests to its hosted objects with probability p. There are n objects in the system. The normalized size of the ith object is s_i with $\sum_{i=1}^{n} s_i = 1$. For systems using single-object requests, we define r_i as the request probability of the ith object, $\sum_{i=1}^{n} r_i = 1$. The number of replicas (including the original copy of the object) used for the ith object is k_i. Hence the total space consumption (normalized by the total data size) is $\sum_{i=1}^{n} s_i \cdot k_i$. We seek to compute object replication degrees, k_i's, that maximize system availability under a space constraint given by $\sum_{i=1}^{n} s_i \cdot k_i \leq K$, where K can be viewed as the normalized total space capacity for replicas. We assume that the underlying replica placement mechanism never assigns two replicas of the same object to one machine. Table 2 lists the symbols defined above.

p	machine failure probability
n	number of objects
k_i	number of replicas used for object i
r_i	request probability of object i
s_i	size of object i (normalized by total data size)
K	total space capacity (normalized by total data size)

Table 2: Symbols used in our simplistic model for availability-oriented object replication.

Given object sizes and popularities, the problem of finding per-object replication degrees that minimizes the expected failure probability of single-object requests can be formalized as the following constrained optimization problem — finding $k_1, k_2, \cdots, k_n$ that

$$\text{Minimize} \sum_{i=1}^{n} r_i \cdot p^{k_i}, \quad \text{subject to} \quad \sum_{i=1}^{n} s_i \cdot k_i \leq K \quad (2)$$

where the optimization goal minimizes the expected request failure probability and the optimization constraint guarantees that the space capacity is not exceeded. The replication degrees $(k_1, k_2, \cdots, k_n)$ must be positive integers in practice. To make our analysis more tractable, we first solve a relaxed version of the optimization problem in which $k_1, k_2, \cdots, k_n$ can be real numbers. We then discuss integer rounding issues at the end of this section. With this relaxation, an optimal solution should always use up all available space (so the optimization constraint becomes $\sum_{i=1}^{n} s_i \cdot k_i = K$).

For the relaxed real-number optimization problem, the solution can be derived using Lagrange multipliers as follows.

As there is just a single constraint, we use only one multiplier λ to combine the constraint and the optimization goal together into the Lagrangian function

$$\Lambda(k_1, k_2, \cdots, k_n, \lambda) = \sum_{i=1}^{n} r_i \cdot p^{k_i} + \lambda \cdot (\sum_{i=1}^{n} s_i \cdot k_i - K) \quad (3)$$

The critical values of Λ is achieved only when its gradient is zero, *i.e.*,

$$\begin{aligned}
&\frac{\partial \Lambda}{\partial k_1} = r_1 \cdot \ln p \cdot p^{k_1} + \lambda \cdot s_1 = 0 \Longrightarrow p^{k_1} = \frac{s_1}{r_1} \cdot \frac{-\lambda}{\ln p} \\
&\frac{\partial \Lambda}{\partial k_2} = r_2 \cdot \ln p \cdot p^{k_2} + \lambda \cdot s_2 = 0 \Longrightarrow p^{k_2} = \frac{s_2}{r_2} \cdot \frac{-\lambda}{\ln p} \\
&\cdots\cdots \\
&\frac{\partial \Lambda}{\partial k_n} = r_n \cdot \ln p \cdot p^{k_n} + \lambda \cdot s_n = 0 \Longrightarrow p^{k_n} = \frac{s_n}{r_n} \cdot \frac{-\lambda}{\ln p} \\
&\frac{\partial \Lambda}{\partial \lambda} = 0 \Longrightarrow \sum_{i=1}^{n} s_i \cdot k_i - K = 0
\end{aligned} \quad (4)$$

Therefore, we know that the optimal solution of $k_1, k_2, \cdots, k_n$ must satisfy

$$p^{k_i} \propto \frac{s_i}{r_i} \Longrightarrow k_i = C + \log_p \frac{s_i}{r_i}, 1 \le i \le n \quad (5)$$

where C is a constant independent of i. Applying the above results to the optimization constraint, we can compute C as follows.

$$\begin{aligned}
&\sum_{i=1}^{n} s_i \cdot k_i = K \\
\Longrightarrow &\sum_{i=1}^{n} \left[s_i \cdot \left(C + \log_p \frac{s_i}{r_i} \right) \right] = K \\
\Longrightarrow &C + \sum_{i=1}^{n} s_i \cdot \log_p \frac{s_i}{r_i} = K \\
\Longrightarrow &C = K + D_{s||r} \cdot \log_p \frac{1}{2}
\end{aligned} \quad (6)$$

where $D_{s||r} = \sum_{i=1}^{n} s_i \cdot \log_2 \frac{s_i}{r_i}$ is the *Kullback-Leibler divergence* [19] between s and r.

Putting (5) and (6) together, we have the solution of k_i's:

$$k_i = K + D_{s||r} \cdot \log_p \frac{1}{2} + \log_{\frac{1}{p}} \frac{r_i}{s_i}, \qquad 1 \le i \le n \quad (7)$$

In our solution, k_i consists of three parts: 1) K, the number of replicas that should be used by uniform replication; 2) $D_{s||r} \cdot \log_p \frac{1}{2}$, a data distribution dependent constant that determines failure probability reduction; 3) $\log_{\frac{1}{p}} \frac{r_i}{s_i}$, an individual deviation that only depends on the properties of object i. Given $p < 1$, this result is consistent with our intuition that an object with a large request popularity and a small size should use a large number of replicas. On the contrary, those objects that have large sizes but small popularities should use few replicas.

The minimized expected request failure probability of our solution is

$$\sum_{i=1}^{n} r_i \cdot p^{k_i} = p^{K + D_{s||r} \cdot \log_p \frac{1}{2}} = p^K \cdot \left(\frac{1}{2} \right)^{D_{s||r}} \quad (8)$$

Note that p^K is exactly the failure probability of uniform replication since it uses $k_1 = k_2 = \cdots = k_n = K$. In comparison, our solution always yields a smaller failure probability ($2^{D_{s||r}}$ times smaller) because $D_{s||r} \ge 0$ is known to hold for all possible distributions of r, s's. In particular, $D_{s||r} = 0$ exactly when s, r are identical, which suggests that uniform replication is optimal exactly when object sizes and popularities follow the same distribution. Uniform distributions of r and s are one obvious special case of this condition. This also suggests that our replication strategy is more beneficial when r, s are more different (hence $D_{s||r}$ is higher). Note that $D_{s||r}$ could be much higher than 1 when s, r are substantially different. For example, if s is uniform and r is a Zipf's distribution, then $D_{s||r} \approx \log_2 \ln n - \frac{1}{\ln 2}$.

Our solution is globally optimal because our optimization goal in (2) is strictly convex[2] and hence its intersection with a hyperplane (the optimization constraint — $\sum_{i=1}^{n} s_i \cdot k_i = K$) is also strictly convex. Consequently, our solution is globally minimal since it is known that a local critical point of a strictly convex function is also its global minimal point.

In order for the above real number solution of $k_1, k_2, \cdots, k_n$ to be used in practice, they have to be rounded into positive integers. We can simply round each k_i to its nearest integer while all k_i's that are smaller than 1 are rounded to 1. Such rounding does not guarantee that the space constraint K is strictly enforced. If a strict enforcement is desired, we can make small adjustments such as rounding down k_i's (as opposed to nearest integer rounding) or scaling down all k_i's before the integer rounding.

4.2 Multi-Object Requests

A multi-object request succeeds only if all involved objects are available. Note that there also exists a loose type of multi-object requests (particularly in the forms of erasure coding [22,23]) in which only a partial subset of the involved objects need to be available. As discussed in Section 2, we view it as an alternative method to whole-object replication for achieving high availability and do not consider it here.

Figure 3 shows that the availability of a multi-object request is affected by different object replica placement. Here we consider two common object placement strategies: random assignment [12,25,34], which randomly assigns objects to machines; partitioning [14, 17, 22], which groups objects into partitions and mirrors each partition on a set of machines.

Random object placement. Consider an m-object request with x_i replicas used for its ith object. Under random placement, the request failure probability is $1 - \prod_{i=1}^{m} (1 - p^{x_i})$, which approximates $\sum_{i=1}^{m} p^{x_i}$ when p^{x_i}'s are small.

The above result suggests that we can view the failure probability of an m-object request as the summed failure probabilities of m independent single-object requests. Therefore, the optimization solution derived in Section 4.1 are directly usable as long as we compute each object (i)'s request probability r_i proportional to the summed probability of all requests that involve object i.

Partitioning-based object placement. Partitioning-based object assignment (PTN) is mainly used when multi-object

[2]A function F is strictly convex if, for all vectors x, y and all $0 < \lambda < 1$, $F(\lambda x + (1-\lambda)y) < \lambda F(x) + (1-\lambda)F(y)$.

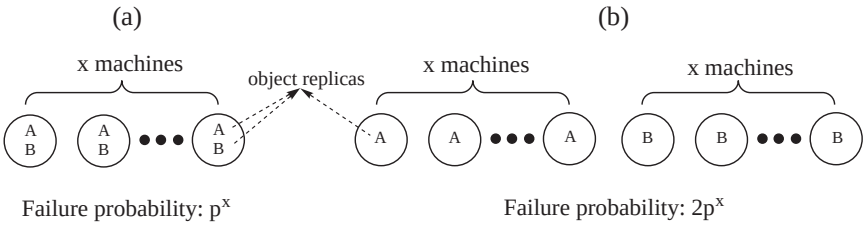

Figure 3: The impact of object placement on the availability of multi-object requests. In the figure, a request accesses both object A and B. Figure (a) assigns A's replicas and B's replicas to the same machines. Figure (b) places A's replicas and B's replicas separately. If each machine fails independently with probability p, then the failure probabilities for the two placements are p^x and $2p^x - p^{2x} \approx 2p^x$, respectively.

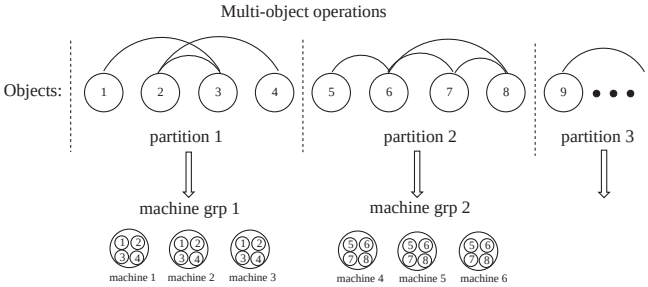

Figure 4: Partitioning-based assignment (PTN) of semantically ordered objects. Objects are partitioned into sets (so that objects within each set tend to be accessed together) and each set is assigned to a group of machines.

access locality exists. By partitioning ordered objects into sets and mirroring each set across a group of machines, PTN realizes that the objects of each multi-object request have the same number of replicas and these replicas are placed on the same group of machines. Figure 4 illustrates such a placement scheme. Due to strong intra-request object correlations, PTN is known to be the best object assignment scheme for multi-object requests [36].

We assume an ideal PTN that always places the object replicas of a multi-object request on the same group of machines, *i.e.*, no cross-group multi-object requests. In this case, viewing each partition as a super-object can turn each original multi-object request into a single-object request on super-objects. Consequently, our basic results for single-object requests are directly applicable for determining the replication degrees of each partition. It is worth noting that the size of a partition is measured in terms of the number of occupied machines since an object partition always exclusively uses a machine as a whole.

Assume the PTN has N object partitions and the total request probability of the multi-object requests on the ith partition is o_i. The total number of machines in the system is S. Every object in the ith partition uses K_i replicas. The optimization problem is to find $K_1, K_2, \cdots, K_N$ that

$$\text{Minimize} \sum_{i=1}^{N} o_i \cdot p^{K_i} \quad \text{subject to} \quad \sum_{i=1}^{N} K_i = S \qquad (9)$$

The optimization goal is the system availability since a multi-object request on the ith partition fails with probability p^{K_i}, independent of the number of objects requested by the request. As illustrated in Figure 4, the ith partition exclusively uses K_i machines. Hence the optimization constraint guarantees that the total number of machines used by all partitions is no more than the system size. Following the approach used in Section 4.1, we can get the solution of K_i's as

$$K_i = \frac{S}{N} + D_{u||o} \cdot \log_p \frac{1}{2} + \log_p \frac{u_i}{o_i}, 1 \le i \le N \qquad (10)$$

where o is the distribution of o_i's and u represents a uniform distribution — $u_i = \frac{1}{N}, 1 \le i \le N$. $D_{u||o}$ is the Kullback-Leibler divergence between o and u. The above result is achieved by viewing each partition as a super-object with K_i replicas, which transforms each original multi-object request into a single-object request on super-objects. Consequently, our results for single-object requests are applicable for determining the optimal replication degrees of each partition. It is also worth noting that the storage consumption of a partition is measured in terms of the number of machines rather than object sizes. This is because an object partition always exclusively uses several machine as a whole.

4.3 Nonuniform Machine Failure Rates

Machines in a distributed system may have nonuniform failure rates due to different hardware/software compositions and configurations as well as varying network accessibility. In such systems, the replica placement can also affect the service availability. For instance, Farsite [10] migrates object replicas between machines with different failure rates to improve availability. However, systems that make such fine-grained replica placement decisions typically require large lookup tables to support placement lookup. On the other hand, scalable placement/lookup schemes like the DHTs [25,28,34] employ random hash-based placement that is oblivious to specific machine properties. In this paper, we assume such a placement scheme in which replicas of each object is placed on machines with no concern on the individual machine failure rates.

We assume that n machines have possibly nonuniform failure probabilities p_1, p_2, $\cdots$, and p_n, respectively. Consider an object with k replicas. When the replica placement is oblivious to individual machine failure rates, the expected object failure probability is $\frac{1}{n^k} \cdot (\sum_{i=1}^{n} p_i)^k$, which is the same as that on uniform failure machines with per-machine failure probability $p = \frac{p_1+p_2+\cdots+p_n}{n}$. This means that our Section 4.1 result is directly applicable here if we set p as the average machine failure probability. Note that the above results do not exclude the possibility that multiple replicas of one object may be assigned to the same machine. If this is

not allowed, the stated result on the expected object failure probability is still approximately true for $k \ll n$.

The above results show that the failure probability of each object changes very little when the nonuniform machines are viewed as uniform machines that fail with the average failure probability. Consequently, the availability of adaptive object replication on nonuniform machines is close to that on corresponding uniform machines.

4.4 Correlated Machine Failures

Machines in a distributed system may not always fail independently. Simultaneous failures are possible due to common failure causes like workload-triggered software bug manifestations and Denial-of-Service attacks [21]. Since the chance of simultaneous replica failures is higher in these cases, strongly correlated machine failures may generally diminish the effectiveness of object replication for high availability. Overall, we believe that our proposed popularity/size-adaptive object replication degree customization would have a reduced but still significant availability enhancement over uniform replication. In particular, our evaluation results in Section 6 show that our availability enhancement is significant except under severe machine failure correlations (*e.g.*, more than 100 simultaneous failures out of 190 machines).

5. ADDITIONAL PRACTICAL ISSUES

Beyond achieving availability, this section addresses a number of issues for applying such a scheme in practice.

5.1 Transparent System Integration

It is desirable to be able to integrate object replication degree customization into an existing replication system without excessive changes. We envision the introduction of a *replication manager* that is responsible for gathering necessary object popularity information and making the replication degree policy. Existing storage machines only need to be slightly changed to collect/report local object access statistics and implement the replication degree policy determined by the replication manager. The availability of the replication manager is not essential for normal system operations since it is only needed for infrequent replication policy adjustments (*e.g.*, weekly or multi-weekly). Nonetheless, the replication manager functionality can be replicated to enhance its availability.

Ideally, the customization of object replication degrees should be transparent to system users. In particular, a user does not need to know the exact object replication degree when accessing an object. For a DHT-compliant random hash-based replica placement scheme, we can place an object (i)'s k_i replicas on machines $hash[f(i,1)]$, $hash[f(i,2)]$, $\cdots$, $hash[f(i,k_i)]$, where $f(\cdot,\cdot)$ can be any deterministic function. Assume we have a system-wide object replication degree upper-bound $k_{\max}$. To look for the object i, a user (with no knowledge of k_i) only needs to check machines $hash[f(i,1)]$, $hash[f(i,2)]$, $\cdots$, $hash[f(i,k_{\max})]$, either serially or simultaneously. In this fashion our proposed object replication degree customization can seamlessly work with any existing object placement scheme that places an object with a specific key to a specific location, *i.e.*, the ith replica of object x is represented by a key tuple [x,i] that solely determine its placement.

5.2 Replica Creation/Deletion under Dynamic System Changes

The replication degree policy may need to be adjusted due to system changes such as varying object request popularities and dynamic machine arrival/departures. A change of the replication degree policy requires replica creations and/or deletions. We consider the overhead of such adjustments below.

Changing object request popularities. We expect that the replica adjustment overhead due to object request popularity changes would not be excessive in practice. First, our analysis of real system object request traces in Section 3.2 suggests that the popularities of most data objects tend to remain stable over multi-week periods. Second, the logarithmic popularity-related term in our proposed replication scheme of Equation (7) infers that exponential changes in object popularity would only call for linear adjustments of the object replication degree. For example, if $p = 0.1$, then ten time popularity increase of an object only amounts to one replica addition. Section 6 will show that no more than 0.24% objects in four evaluated real-life application workloads need weekly replica adjustment. The popularity changes of other objects are not high enough to cause changes in the number of their replicas.

Furthermore, we have the option of lowering the replica adjustment overhead by using a coarser policy granularity when determining object replication degrees. For example, Figure 5(A) uses a fine-grained policy where the replication degree can be any positive integers. In contrast, Figure 5(B) employs a coarser policy granularity and hence it is less sensitive to object popularity changes. Note that this scheme may lead to diminished availability gain at the same time.

For the purpose of determining the thresholds of a coarse-grained policy, the knee points on the curve of object popularity-to-size ratios may serve as good candidates. The final choice among such candidates is subject to further experiments.

Dynamic machine arrival/departures. Dynamic machine arrival/departures typically change the total space capacity K in the system and as a result the object replication degrees may need to be adjusted. According to our proposed replication scheme in Equation (7), a change of K requires a uniform adjustment of replication degrees for all objects. Note that such adjustment is identical to that of uniform replication. Therefore our customized replication scheme incurs no extra replica adjustment overhead over uniform replication. Such adjustment overhead should be small when the space capacity K does not change significantly.

5.3 Replica Consistency Maintenance

Some applications may allow write requests on the data objects. Writes incur replica consistency maintenance overhead and the overhead grows as more replicas are used. In a simple read-one/write-all protocol, the object write overhead is linear in the number of object replicas. For Byzantine fault tolerance protocols [4], Kubiatowicz *et al.* [18] pointed out that a nearly linear overhead is achievable if the the replication degree is not too large. Generally speaking, by using more replicas for popular objects, popularity-adaptive replication degree customization tends to incur more replica consistency maintenance cost than uniform replica-

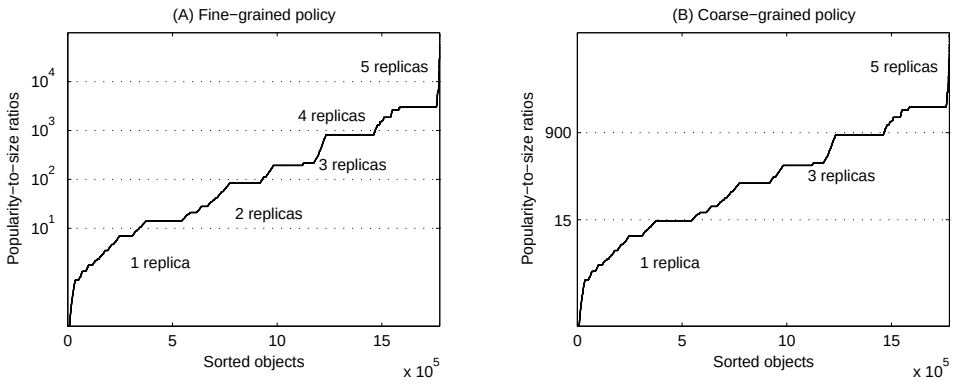

Figure 5: An example of fine-grained and coarse-grained policies for determining the object replication degrees of popularity/size-adaptive replication. The popularity-to-size ratio is defined as $\frac{r_i}{s_i}$ for object i.

tion. Such extra overhead can be bounded by enforcing an upper-bound on per-object replication degrees ($k_{\max}$). We will quantitatively evaluate such overhead in Section 6.

6. EVALUATION

This section evaluates the effectiveness and feasibility of popularity/size-adaptive object replication on data-intensive applications driven by real-life traces. Our evaluation considers the impact of a wide range of practical factors, including single/multi-object requests, object placement, failure patterns (uniform/nonuniform, independent/correlated), dynamic object popularity changes, and replica consistency maintenance overhead.

6.1 Evaluation Settings

Our evaluation framework supports different replication strategies and also allows the injection of various workloads and failure patterns.

Object replication degree policies. We consider uniform replication and two customized policies at different rounding granularities (with performance/overhead trade-off). The comparison of the three policies is made at the same space consumption — when the average object replication degree (weighted by object sizes) is the same.

- *Uniform.* All objects use the same number of replicas.
- *Fine-grained object replication degree customization.* The replication degree of object i is determined by rounding k_i (7) to the nearest integer within $[1, k_{\max}]$, where $k_{\max}$ sets an upper-bound for per-object replica consistency maintenance cost. In our evaluation, we set $k_{\max}$ as four times the average replication degree over all objects (weighted by object sizes).
- *Coarse-grained object replication degree customization.* This policy is the same as the fine-grained customization scheme except that k_i is rounded in a different way. In this policy, there are only two possible replication degrees (high and low) for each object. Each object chooses from the two replication degrees based on its popularity-to-size ratio ($\frac{r_i}{s_i}$ for object i). More specifically, an object uses the high replication degree if its popularity-to-size ratio is higher than a threshold and it uses the low replication degree otherwise. The threshold is determined at one of the knee points on the object popularity-to-size ratio curve, as explained in Section 5.2. The high and low replication degrees are carefully selected such that the space constraint is not violated. Compared with fine-grained replication degree customization, this scheme incurs less replica adjustment overhead in response to object popularity changes, but at the cost of less availability.

Workloads. Our evaluation employs four data-intensive workloads driven by real system data object request traces.

- *Distributed web store.* We collect a web page request trace from the NLANR web caching project (IRCache) [15], which consists of ten cooperative web caches. The trace spans over six weeks of 09/15/2005–10/27/2005. We use the trace to drive a distributed web content store application in which each data object is a web page. Due to a legacy trace processing artifact (for the purpose of caching), our processed trace does not contain those web pages that are only accessed once in the six-week duration (since extremely rare web pages serve no purpose of caching). This workload contains 4.89 million distinct web pages and 33.46 million requests.
- *Index search.* We build a prototype web search engine. The search engine allows full-text keyword search on 3.7 million web pages. The web pages were crawled following URL listings of the Open Directory Project [9] and preprocessed by using the stopword list of the SMART software package [33]. The queries are from a partial query log of 6.8 million queries at the Ask.com search engine [1]. There are an average of 2.54 keyword terms per query. In our search application, a data object is the data index associated with each keyword. A multi-keyword search accesses the data indices associated with all keywords in the query.

  This workload allows multi-object requests in the form of multi-keyword queries. Since the object placement policy may affect the system availability, we consider

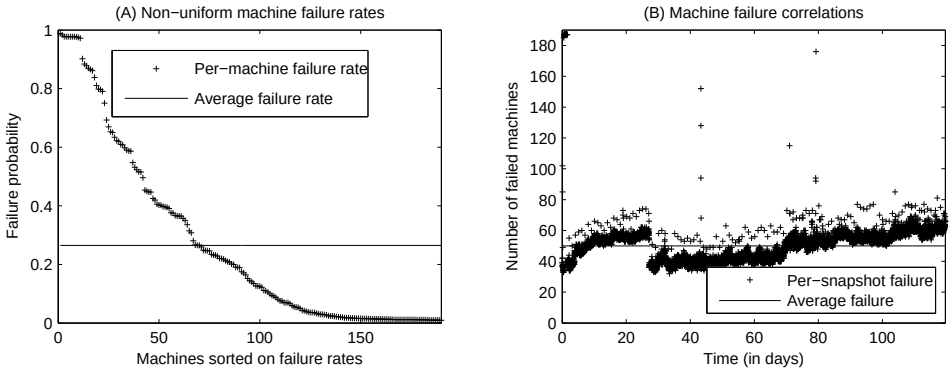

Figure 6: The PlanetLab machine failure probabilities and failure correlations. The points that are well above the average in Figure (B) represent correlated machine failures.

two common object placement policies that we discussed in Section 4.2: Random (or RAND) and PTN. We thus have two variations of this workload. The object partitions in our implementation of PTN are generated by using the Chaco [5] software package to divide query keywords into semantically clustered groups — the keywords in each group tend to be queries together while there are few cross-group queries. An example of such keyword groups include the following words: `san`, `ca`, `diego`, `francisco`, `puerto`, `tx`, `austin`, `rico`, `antonio`, `earthquake`, `jose`, `juan`, `vallarta`, `lucas`, `rican`, `luis`, `cabo`, `fransisco`, `bernardino`.

- *Distributed file store (Harvard).* We use the Harvard NFS trace [11] over six weeks (02/01/2003–03/14/2003) to drive a distributed file store application. The raw data trace contains all NFS events over the network. Note that a single file access session from a client on a file may consist of many NFS file access events. We believe it is more appropriate to consider these NFS events as one data object request since they belong to a single client file access session and they share the same success/failure outcome depending on the object (file) availability. Specifically, we merge multiple file access events from the same client on the same file over a certain time period (*e.g.*, one hour) into a single request. The processed trace contains 2.78 million requests on 667,656 files.

- *Distributed file store (Coda).* We employ another distributed file store workload driven by a Coda distributed file system [17] trace over one month. The trace contains 556,703 requests on 16,250 files.

The IRCache and Ask.com search workloads are both read-only. On the other hand, the Harvard and Coda distributed file system workloads contain some writes. Specifically, the write ratio for the Coda trace is less than 6% while the write ratio in the Harvard NFS trace is around 37%. Our evaluation on availability only considers the read portions of these workloads. We will examine the replica consistency maintenance overhead due to writes in Section 6.3.

Failure patterns. Our availability evaluation uses two real-life distributed system machine failure patterns (an Ask.com LAN service trace and a PlanetLab WAN service trace) and one synthetic WAN failure pattern.

- *A six-month Ask.com local-area cluster failure trace.* The trace concerns a cluster of 300 machines with an MTTF (mean time to next failure) of 275 days and an MTTR (mean time to next repair) of around 3 hours. The average machine failure probability is $p = 0.000455$. Such a lower failure rate is typical for centrally-managed local-area clusters.

- *A four-month PlanetLab node accessibility trace.* The trace contains all-pair ping snapshots at 20-minute intervals for some PlanetLab nodes over the period of June–September 2006. We consider a node fails at a snapshot if no other nodes can successfully ping it. There are 190 nodes left after we remove those nodes that never come alive over the four-month period. The average node failure probability is $p = 0.265$. This trace possesses a highly skewed machine failure probability distribution (Figure 6(A)) and strong failure correlations (Figure 6(B)).

- *A synthetic failure pattern for WAN services.* The PlanetLab testbed probably suffers from higher-than-normal failures due to its nature as an experimental research platform. To consider more general patterns, we also utilize a synthetic WAN service failure model derived from network connectivity and HTTP server availability [8]. Specifically, this failure model consists of independent machines each with an MTTF of 11571 seconds and an MTTR of 609 seconds. Each machine fails with probability $p = 0.05$.

System setup. In our experiment, the object request popularity distribution is available to the system a priori. The distribution is updated on a weekly basis and we adjust our popularity-aware object replication degrees accordingly. We run the object request workloads on the simulated nodes, which fail by strictly following the observed failure time and durations in the failure trace.

We measure system availability in the commonly used notation of "nines". For a system with failure probability of f, its "number of nines" is defined as $\log_{0.1} f$. For example, a system with 3-"nines" availability means that

	Web store	Index search RAND	Index search PTN	File store (Harvard)	File store (Coda)
Coarse-grain	1.73 nines	1.18 nines	1.14 nines	1.07 nines	1.82 nines
Fine-grain	2.12 nines	1.53 nines	1.32 nines	1.71 nines	2.24 nines

Table 3: The availability gain of customized replications over uniform replication on the Ask.com failure pattern. Due to the low average machine failure rate (0.000455), we focus on a low average object replication degree for uniform replication — 2 replicas per object. Our customized replications use the same amount of total space as uniform replication.

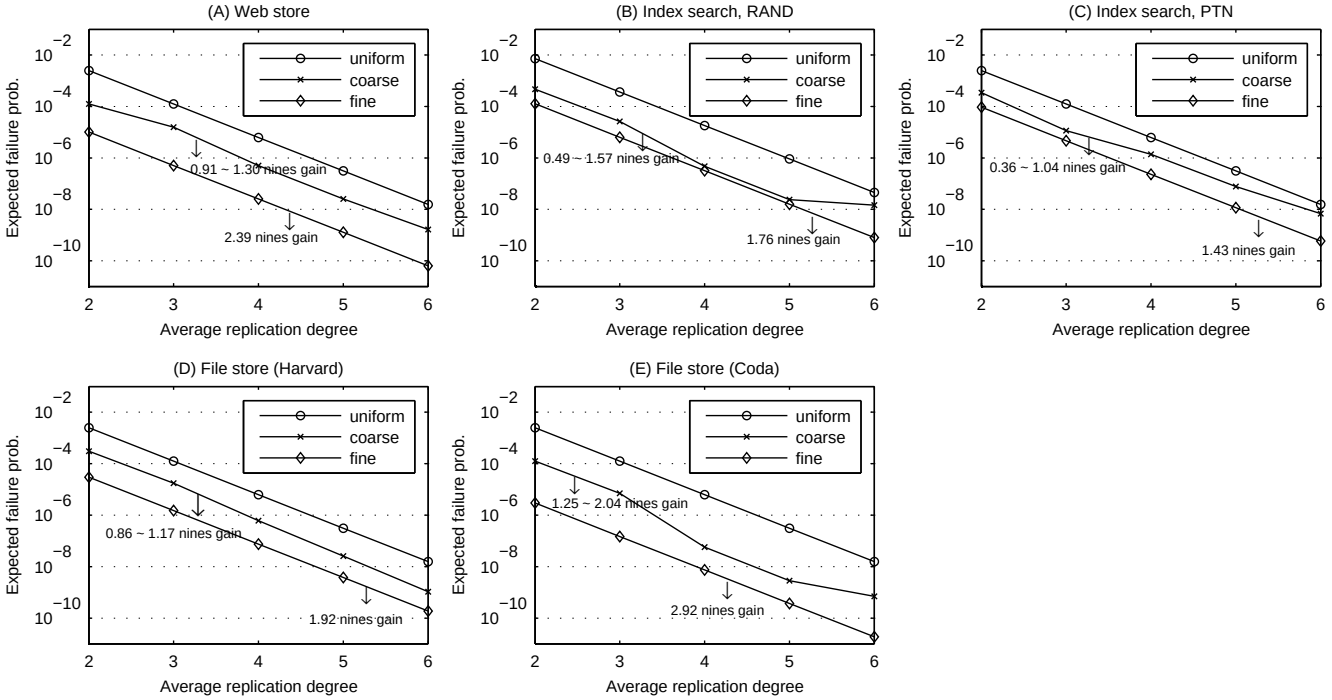

Figure 7: The failure probabilities and availability gain on the synthetic WAN service failure pattern (uniform independent machine failure rate $p = 0.05$). The marked availability gains are over the uniform replication. All three replication strategies use the same amount of total space.

the system is available for 99.9% time. By saying that system A (failure probability: f_A) achieves x "nines" availability gain over system B (failure probability: f_B), we mean that $\log_{0.1} f_A - \log_{0.1} f_B = x$.

6.2 Effectiveness

Table 3 shows the effectiveness of our proposed replication degree customization on the Ask.com machine failure pattern. Due to the low average machine failure rate, we focus on a low average object replication degree (2 replicas per object) for this failure pattern. Results show that the proposed popularity/size-adaptive replication achieves up 1.32–2.24 nines availability increase over uniform replication. Even the coarse-grained adaptive scheme (that only allows two distinct replication degrees) achieves 1.07–1.82 nines availability enhancement.

Figure 7 studies the failure probabilities on the synthetic WAN service failure pattern. We show the results on varying average replication degrees. The results show that fine-grained replication degree customization achieves 1.43–2.92 nines availability increase over uniform replication. Furthermore, the increase remains stable over different average replication degrees. Such stability matches our analytical result in Equation (8), where the failure probability reduction factor ($2^{D_{s||r}}$) only depends on data distributions. We notice that the availability increase of coarse-grained replication degree customization may vary in different settings. We believe this is due to relative instability of this scheme.

Figure 8 studies the availability gain of the proposed replication degree customization on the PlanetLab machine failure pattern, which possesses a highly skewed machine failure probability distribution and strong failure correlations (Figure 6). Results show that the fine-grained customization achieves 1.38–2.34 nines availability increase over uniform replication, which are slightly inferior to the gains on the synthetic WAN load with uniform independent machine failures.

We performed further study to examine the impact of correlated machine failures on the availability gain of our proposed popularity/size-adaptive replication. As there is no

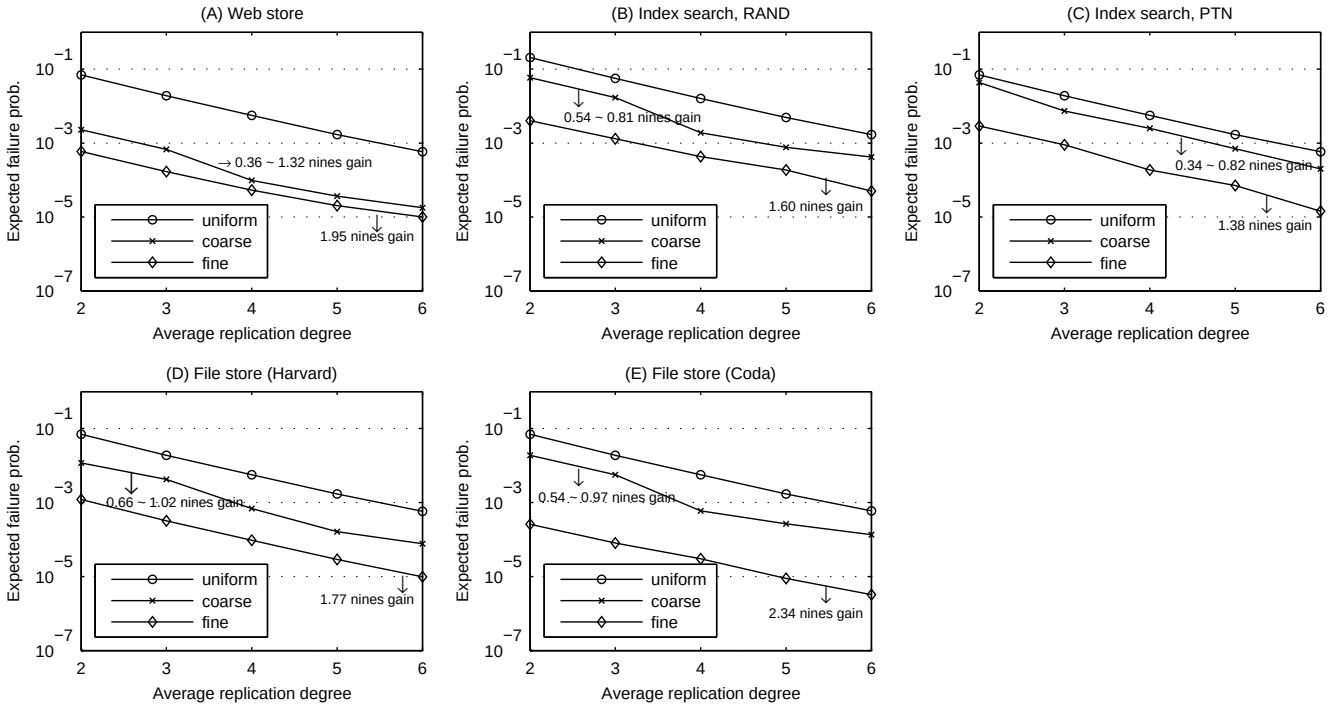

Figure 8: The failure probabilities and availability gain on the PlanetLab failure pattern (with highly correlated machine failures). The marked availability gains are over the uniform replication. All three replication strategies use the same amount of total space.

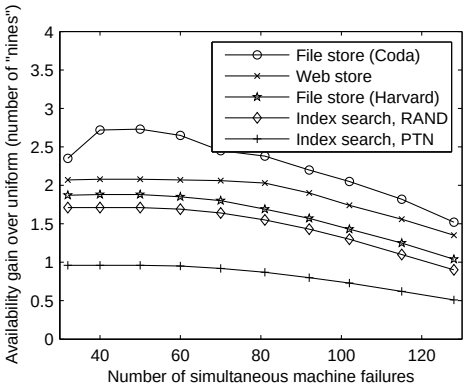

Figure 9: The impact of correlated machine failures on the availability gain.

clear boundary to differentiate correlated machine failures and non-correlated failures purely by looking at the failure trace, we view a lot of simultaneous machine failures as likely correlated failures and study their impact. The results in Figure 9 are based on the PlanetLab failure trace. It shows that strongly correlated failures (many simultaneous machine failures) indeed lead to diminished availability gains but the reduction is not significant except under severe machine failure correlations (*e.g.*, $\geq$ 100 simultaneous failures out of 190 machines).

In Table 3, Figure 7, and Figure 8, popularity/size-adaptive object replication degree customization achieves 1.32–2.92 nines availability gain over uniform replication with the same space consumption (or 21–74% space savings at the same availability level). Furthermore, the availability gain mainly depends on the difference between the object popularity distribution and size distribution — the higher the difference, the more the availability gain. Specifically on our studied four workloads (one of which has two variations), the availability gain follows the order of "Index search, PTN" < "Index search, RAND" < "File store (Harvard)" < "Web store" < "File store (Coda)". This can be explained by characteristics of respective workload driving traces. As shown in the lower row of Figure 1, the size-normalized object popularity skewness of the traces follows the order of "Ask.com" < "Harvard NFS" < "IRCache" < "Coda". Note that the size normalization is important. Particularly for the Ask.com trace, there is a high skewness in object request popularities but the skewness is much lower for the size-normalized popularities. This is because a popular keyword tends to be associated with a large data index (the list of web pages containing the keyword).

6.3 Feasibility

Beyond achieving high availability, the practical application of popularity/size-adaptive object replication degree customization must address issues including dynamic replica adjustment overhead (caused by object popularity changes) and increased replica consistency maintenance overhead (due to higher replication degrees for more popular objects). Here we evaluate them quantitatively.

The popularity of individual data objects may not stay constant over the lifetime of a data-intensive service. In

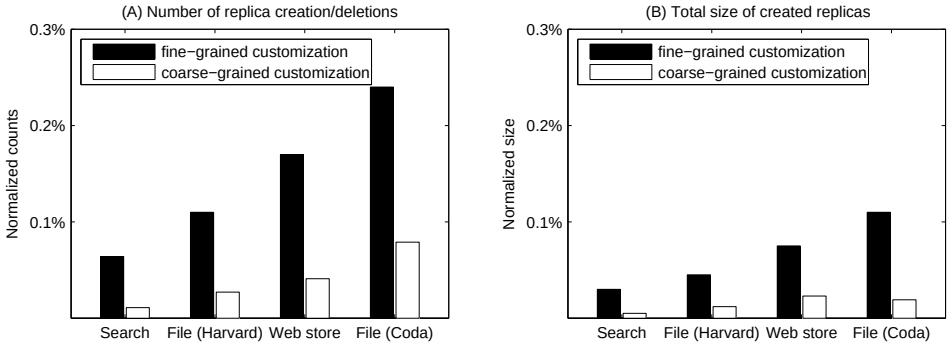

Figure 10: Weekly replica adjustment overhead caused by dynamic object popularity changes on the PlanetLab failure pattern. Results are normalized to the total number of objects or the total data size. The overhead is even lower on other two failure patterns due to their smaller average machine failure rates.

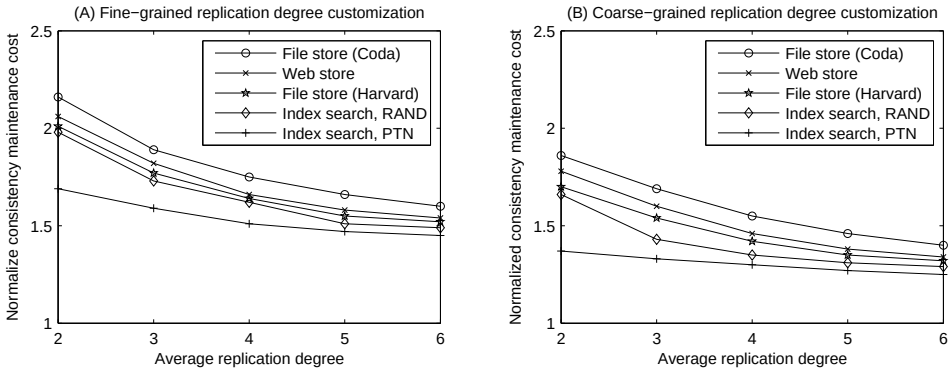

Figure 11: Replica consistency maintenance overhead of customized replication (normalized by the overhead of uniform replication).

the extreme case, some objects may only become popular for short bursts of time (*e.g.*, breaking news and security patches for short-lived viruses). Our object replication degree customization is not intended to adapt to such dramatic object popularity changes (flash crowds) due to large cost of replication degree adjustment. Instead, we only make replication degree adjustment in response to long-term stable changes of object popularities. Fortunately our analysis of four real service traces in Section 3.2 suggests that the object popularity in many large-scale services tend to be stable over multi-week periods.

Based on our four workloads driven by real-life traces, Figure 10 shows that the weekly adjustment overhead is low in relation to the total number of objects and the total data size. This is because the optimal object replication degree ($C+\log_{\frac{1}{p}} \frac{r_i}{s_i}$ for object i) is fundamentally insensitive to object popularity changes — $\frac{1}{p}$ times increase in the popularity of an object only leads to one more replica. Consequently, no excessive adjustment is needed unless the change is very substantial. Furthermore, the overhead becomes much smaller when the coarse-grained adaptive replication is used.

In practical systems, the number of machines may also fluctuate due to dynamic machine arrivals or departures. These events affect available space capacity, which may consequently desire adjustments on object replication degrees. However, our customized replication scheme does not require more adjustments of this type than uniform replication does, as explained at the end of Section 5.2. Therefore we do not quantitatively evaluate such overhead here.

By using more replicas for popular objects, our customized replication tends to incur more replica consistency maintenance cost for object writes than uniform replication does. In our evaluation, we assume that object write overhead is linear in the number of object replicas. This is true for simple read-one/write-all protocols and approximately true for Byzantine fault tolerance protocols as pointed out by Kubiatowicz *et al.* [18]. Assuming that object updates follow the same popularity distribution as object requests, Figure 11 shows that the average object update overhead of fine-grained customization is at most 2.2 times that of uniform replication. The coarse-grained customization incurs a smaller overhead (no more than 1.8 times that of uniform replication). Although such overhead increase is substantial, the replica consistency maintenance overhead is only incurred for object mutations and thus it is not a significant concern for read-only or read-mostly services.

7. CONCLUSION

In this paper, we study the problem of achieving high service availability under given space constraints by customizing object replication degrees to object popularities and sizes. Such object popularities are highly skewed in

typical distributed data-intensive applications. We find that the optimal replication degree for an object is linear in the logarithm of its popularity-to-size ratio. We further address a number of practical issues in deploying our proposed customization, including multi-object requests, object placement, nonuniform machine failure rates and correlated failures, dynamic object popularity changes, and increased replica consistency maintenance cost. Our quantitative evaluation employs workloads driven by large real-life system data object request traces (IRCache web proxy cache [15] request trace, Ask.com [1] web keyword search trace, Harvard NFS file access trace [11], and CMU Coda file system [17] access trace) and machine failure traces in real distributed systems (Ask.com server cluster and PlanetLab machines).

Our main results on its effectiveness are:

- The effectiveness of popularity/size-adaptive replication mainly depends on the difference between the object popularity distribution and size distribution — the higher the difference, the more the availability gain. On our four evaluated workloads, popularity/size-adaptive replication achieves 1.32–2.92 nines availability gain over uniform replication with the same space consumption (or 21–74% space savings at the same availability level).

- The availability gain of our popularity/size-adaptive replication strategy does not change substantially over different space constraints, individual machine failure rates, and moderately correlated machine failures (*e.g.*, fewer than 100 failures out of 190 PlanetLab machines).

- Compared to the standard form of our proposed replication degree customization, coarse-grained customization that dichotomizes objects based on their popularity-to-size ratios can reduce the replica adjustment overhead caused by object popularity changes. Specifically, a 3.1–6.0 times overhead reduction is achieved by a particular coarse-grained scheme introduced in this paper. At the same time, its availability gain (0.18–1.80 nines) is lower than the standard (fine-grained) replication degree customization.

Our main results on its feasibility are:

- The replica adjustment overhead caused by dynamic object popularity changes is not high in practice, *e.g.*, the weekly replica adjustment is no more than 0.24% of the total number of objects and no more than 0.11% of the total object size on our studied workloads.

- For object writes, the extra replica consistency maintenance overhead caused by popularity-adaptive replication can be quite high. Specifically, the overhead of popularity-adaptive replication is up to 2.2 times that of uniform replication. However, this overhead is only incurred for write requests and thus it is not a significant concern for read-only or read-mostly services.

Based on our results, we conclude that popularity/size-adaptive object replication degree customization is beneficial for improving system availability under space constraints when 1) the application is not write-intensive, 2) the object popularities follow a highly skewed distribution, and 3) the object size distribution is significantly different from the object popularity distribution. Many data-intensive application workloads satisfy the above conditions. However, when deploying customized replication, one must be careful about periodic replica adjustment overhead (caused by dynamic object popularity changes) and increased replica consistency maintenance overhead. Such overhead can be controlled by tuning the granularity for determining object replication degrees — lower overhead can be achieved by a coarser-grained customization at the cost of weaker availability gain.

Acknowledgments

We thank Lingkun Chu and Tao Yang at Ask.com for providing us the index search dataset and query traces, as well as for giving us a six-month local-area cluster failure trace used in this study. We thank Jonathan Ledlie at Harvard University for providing us the NFS network event trace. We also thank Pin Lu at the University of Rochester for helping the transfer of the Harvard NFS trace. Finally, we would like to thank the EuroSys reviewers for their valuable comments that helped improve this paper.

8. REFERENCES

[1] Ask.com, formerly Ask Jeeves Search. http://www.ask.com.

[2] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G.M. Voelker. Total recall: System support for automated availability management. In *Proc. of the First USENIX Symp. on Networked Systems Design and Implementation*, pages 337–350, San Francisco, CA, March 2004.

[3] W.J. Bolosky, J.R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PC. In *Proc. of the ACM SIGMETRICS*, pages 34–43, Santa Clara, CA, June 2000.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the Third USENIX Symp. on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

[5] Chaco. http://www.cs.sandia.gov/∼bahendr/chaco.html.

[6] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. of the ACM SIGCOMM*, pages 177–190, Pittsburgh, PA, August 2002.

[7] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, pages 202–215, Banff, Canada, October 2001.

[8] M. Dahlin, B. Chandra, L. Gao, and A. Nayate. End-to-end WAN service availability. *ACM/IEEE Tran. on Networking*, 11(2):300–313, April 2003.

[9] The open directory project. http://www.dmoz.com.

[10] J.R. Douceur and R.P. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proc. of the 15th Int'l Symp. on Distributed Computing*, pages 48–62, Lisboa, Portugal, October 2001.

[11] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads.

In *Proc. of the Second USENIX Conf. on File and Storage Technologies*, San Francisco, CA, April 2003.

[12] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. In *Proc. of the 19th ACM Symp. on Operating Systems Principles*, pages 29–43, Bolton Landing, NY, October 2003.

[13] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for internet service construction. In *Proc. of the 4th USENIX Symp. on Operating Systems Design and Implementation*, San Diego, CA, October 2000.

[14] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proc. of the Second USENIX Symp. on Networked Systems Design and Implementation*, pages 143–158, Boston, MA, May 2005.

[15] IRCache. http://www.ircache.net.

[16] J. Kangasharju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. In *Proc. of the 6th Workshop on Web Caching and Content Distribution*, Boston, MA, June 2001.

[17] J.J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Trans. on Computer Systems*, 10(1):3–25, February 1992.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of the 9th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, Cambridge, MA, November 2000.

[19] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[20] J.W. Mickens and B.D. Noble. Exploiting availability prediction in distributed systems. In *Proc. of the Third USENIX Symp. on Networked Systems Design and Implementation*, 2006.

[21] S. Nath, H. Yu, P.B. Gibbons, and S. Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *Proc. of the Third USENIX Symp. on Networked Systems Design and Implementation*, pages 225–238, San Jose, CA, May 2006.

[22] D.A. Patterson, G. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the ACM Int'l Conf. on Management of Data*, pages 109–116, Chicago, IL, September 1988.

[23] J. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software Practice and Experience*, 27(9):995–1012, September 1997.

[24] V. Ramasubramanian and E. Sirer. Beehive: Exploiting power law query distributions for o(1) lookup performance in peer to peer overlays. In *Proc. of the First USENIX Symp. on Networked Systems Design and Implementation*, San Francisco, CA, March 2004.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM*, pages 161–172, San Diego, CA, August 2001.

[26] R.V. Renesse and F.B. Schneider. Chain replication for supporting high throughput and availability. In *Proc. of the 6th USENIX Symp. on Operating Systems Design and Implementation*, pages 91–104, San Francisco, CA, December 2004.

[27] R. Rodrigues and B. Liskov. High availability in DHTs: Erasure coding vs. replication. In *Proc. of the 4th Int'l Workshop on Peer-to-Peer Systems*, Ithaca, NY, February 2005.

[28] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware Conf.*, pages 329–350, Heidelberg, Germany, November 2001.

[29] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, availability, and performance in Porcupine: a highly scalable, cluster-based mail service. In *Proc. of the 17th ACM Symp. on Operating Systems Principles*, pages 1–15, Charleston, SC, December 1999.

[30] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated Resource Management for Cluster-based Internet Services. In *Proc. of the 5th USENIX Symp. on Operating Systems Design and Implementation*, pages 225–238, Boston, MA, December 2002.

[31] K. Shen, T. Yang, and L. Chu. Clustering support and replication management for scalable network services. *IEEE Trans. on Parallel and Distributed Systems*, 14(11):1168–1179, November 2003.

[32] K. Shen, T. Yang, L. Chu, J. L. Holliday, D. A. Kuschner, and H. Zhu. Neptune: Scalable replication management and programming support for cluster-based network services. In *Proc. of the Third USENIX Symp. on Internet Technologies and Systems*, pages 197–208, San Francisco, CA, March 2001.

[33] SMART. ftp://ftp.cs.cornell.edu/pub/smart.

[34] 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. of the ACM SIGCOMM*, pages 149–160, San Diego, CA, August 2001.

[35] H. Weatherspoon and J.D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of the First Int'l Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[36] H. Yu, P.B. Gibbons, and S. Nath. Availability of multi-object operations. In *Proc. of the Third USENIX Symp. on Networked Systems Design and Implementation*, pages 211–224, San Jose, CA, May 2006.

[37] H. Yu and A. Vahdat. Minimal replication cost for availability. In *Proc. of the 21st ACM Symp. on Principles of Distributed Computing*, pages 98–107, Monterey, CA, July 2002.