Storage-Efficient Data Replica Number Computation for Multi-level Priority Data in Distributed Storage Systems

Chris X. Cai, Cristina L. Abad*, Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
{xiaocai2,cabad,rhc}@illinois.edu

***Abstract*—Distributed storage systems often use replication for improved availability, performance and scalability. In this paper, we consider the case of using file replication to improve the availability of different classes of files, where some classes are more "important" than others and more replicas are created for them to achieve improved availability. The question we attempt to answer is: given a fixed storage budget for storing replicas, what is the number of replicas of each file class to create to maximize the (weighted) overall availability of files? We present our work towards a replica number computation algorithm that takes into account a storage budget, a configurable maximum expected percentage of failed nodes, and weights for different file classes. Simulation results show that our algorithm is able to improve the availability of the prioritized files with higher weights, has a low computation time and can utilize storage space efficiently when total storage space scales to a large size.**

***Index Terms*—storage system, replication, availability, optimization, budget**

I. Introduction

Distributed storage systems are increasingly used to store Big Data in Internet services companies like Google, Facebook and Yahoo!. Examples of distributed storage systems include: Google File System [1], Hadoop Distributed File System (HDFS) [2] and Ceph [3]. These systems frequently use data replication—in which multiple copies of a file are replicated across a certain number of data nodes—to improve data availability, performance and scalability [1, 2, 4]. In the presence of data node failures, a file is still available as long as at least one copy is reachable.

Storage systems that support replication allow administrators or users to change the replication factor of files, or at some other granularity like per block or per directory. For example, by default HDFS creates three replicas of each file but allows users to manually change the replication factor of a file. In these systems, manually increasing or decreasing the number of replicas for a file using organization heuristics (e.g., based on data access patterns) is a common approach. For example, Facebook de-replicates aged data, which can have a lower number of replicas (as low as one copy) compared to other data [5]. However, as next-generation storage systems are built for increasingly larger storage capacities, autonomic approaches to replication are desirable. Ideally, a set of replication mechanisms or policies is available for users to choose from, according to their specific requirements [6].

Several automated replication mechanisms have been proposed, each with different goals like improved job performance (when data is co-located with computation) [7], achieving a target availability [8], etc. In this paper, we consider the case of maximizing the (weighted) *overall availability* of files given a replication budget and a set of file class weights. We propose CaB[1], an autonomic replication number computation algorithm that assigns more replicas for the files belonging to highest priority classes and less replicas for files in lower priority classes, without exceeding the replication budget.

CaB can be useful in several scenarios. For example, consider the case of a cloud storage provider that offers several storage plans with different services and guarantees. While it could provide strong SLA-based reliability guarantees to premium customers, it could also offer best-effort budget-based replication to entry-level customers. With this second approach, the client could explicitly choose her storage budget (under some pricing scheme) to be used to store replicas for her files, based on their priority or availability class. These priorities could be user-defined or workload-based (based on file access patterns).

We compare CaB with a baseline algorithm that assign storage space to files proportional to their priority weights. Simulation results show CaB is able to improve the availability of the prioritized files with higher weight. Furthermore, compared to the baseline algorithm, CaB utilizes storage space more efficiently when the total storage is large. Additonally, CaB has a low computation time to run, making it suitable for periodic recalculation of replica number assignments.

The rest of this paper is structured as follows. In § II we describe our system model, including assumptions and simplifications. In § III we describe our *overall availability* metric and describe CaB, an algorithm for computing the number of data replicas per class of files so that the *overall availability* is maximized. We present some preliminary evaluation results in § IV. Finally, we discuss our future work (§ V) and conclude (§ VI).

II. System Model

We consider a storage system in which files are divided into fixed-sized blocks, each of which is stored and replicated independently of the other blocks in the file. GFS [1] and HDFS [2] are two examples of storage systems that use this approach. For example, by default HDFS divides files blocks;

* Also affiliated with Facultad de Ingeniería en Electricidad y Computación (FIEC), Escuela Superior Politécnica del Litoral (ESPOL), Campus Gustavo Galindo, Km 30.5 Vía Perimetral, Guayaquil–Ecuador.

[1]CaB: **C**lass **a**nd **B**udget-based replica number computation algorithm.

978-1-4799-0181-4/13/$31.00 ©2013 IEEE

the block size is configurable, and is set to 128 MB by default. In our model, each file is composed of n file blocks $\{b_1, b_2, b_3, ..., b_n\}$. A file block is unavailable if all of the replicas of that file block are unavailable, e.g., all of the storage nodes which store the replicas of that file block are unavailable.

We use a number between 0 and 1 to represent the *weight* of file. The higher *weight* of a file has, the more priority our data replication scheme will give it when storage space is available for data replicas. We define a *file class* to be a class of files which have the same *weight*. We denote a file class to be C_i, the *weight* associated with that class to be W_i, and the number of files in a file class to be N_i.

We model a distributed storage system to have a number of storage nodes in a homogeneous cluster, where each node has same storage capacity. We assume each storage node has the same expected failure probability $\bar{f}$; node failure is assumed to be random and independent. The system has T storage nodes. Each file class has associated number of replicas r_i and partial replicas p_i, which is number of files in a file class assigned one more replica due to insufficient storage to accommodate all files. Table I shows a complete list of model parameters and definitions.

Our current model does not take time into account. The expected failure probability of a node, $\bar{f}$, is the probability that a node fails some between t_0 (now) and some specific time in the future t_1 (e.g., in 24 hours). In § V we discuss how to relax this assumption.

TABLE I
MODEL PARAMETERS AND DESCRIPTIONS

Parameter	Description
FC_i	file class i
W_i	class weight associated with C_i
N_i	number of files in C_i
B_{ij}	number of file blocks in j^{th} file in C_i
m	number of file classes
T	total number of storage nodes
S	total available storage budget
r_i	number of replicas assigned to C_i
$rspace_i$	remainder space assigned to C_i

III. DATA REPLICA COMPUTATION ALGORITHM

A. Availability Metric

Since our model considers heterogeneous types of data stored in the system, we need an availability metric capable of capturing the different priorities of data. We propose the following metric:

$$OverallAvailability = \frac{\sum_{i=1}^{m} W_i N_i A_i}{\sum_{i=1}^{m} W_i N_i} \tag{1}$$

In Equation (1), i iterates over each file class FC_i. A_i is the availability of file class FC_i. Both the *weight* and file number N_i of a file class affect the contribution of the class availability A_i to the overall availability.

B. Computing Number of Data Replicas

Our goal is to fully utilize the storage space and achieve the highest availability according to the availability metric in § III-A. We want to compute the ideal number of data replicas for each file class given a storage space constraint. Let S be the total storage space given in number of blocks, T be the number of storage nodes available, and r_i be the number of replicas we store for file class FC_i. Our problem formulation can be expressed as:

$$\begin{aligned} \underset{r_i}{\text{maximize}} \quad & \frac{\sum_{i=1}^{m} W_i N_i A_i}{\sum_{i=1}^{m} W_i N_i} \\ \text{subject to} \quad & \sum_{i=1}^{m}(r_i \sum_{j=1}^{N_i} B_{ij}) \leq S, \quad \forall i \quad r_i \leq T \quad r_i \in \mathbb{N} \end{aligned} \tag{2}$$

The number of replicas we assign to each file class should be less or equal to the number of storage nodes; storing more than one replica on a single node does not improve the availability of a file.

If we assume node failures are independent, then A_i in Equation (2) can be calculated as follows.

Let $\bar{f}$ be the maximum expected percentage of storage nodes that fail sometime during t_o, t_1; then, the probability that all replicas of a block of a file in file class FC_i are unavailable at t_1 is:

$$\bar{f}^{r_i}$$

For j^{th} file in file class FC_i which has B_{ij} file blocks, the probability of at least one block being unavailable at t_1 can be calculated as follows, which is derived from [8], where $C^k_{B_{ij}}$ is the number of k-combinations from a B_{ij}-element set:

$$\sum_{k=1}^{B_{ij}} (-1)^{k+1} C^k_{B_{ij}} (\bar{f})^{kr_i}$$

Then, the probability that at least one replica of each file block of j^{th} file in class FC_i is still available at t_1 is:

$$1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C^k_{B_{ij}} (\bar{f})^{kr_i} \tag{3}$$

The fraction of available files in FC_i which has N_i files at t_1 is:

$$\frac{\sum_{j=1}^{N_i}(1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C^k_{B_{ij}} (\bar{f})^{kr_i})}{N_i} \tag{4}$$

Substituting A_i in (2) with the availability calculated in Equation (4) gives:

$$\begin{aligned} \underset{r_i}{\text{maximize}} \quad & \frac{\sum_{i=1}^{m} W_i \sum_{j=1}^{N_i}(1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C^k_{B_{ij}} (\bar{f})^{kr_i})}{\sum_{i=1}^{m} W_i N_i} \\ \text{subject to} \quad & \sum_{i=1}^{m}(r_i \sum_{j=1}^{N_i} B_{ij}) \leq S, \quad \forall i \;\; r_i \leq T, r_i \in \mathbb{N} \end{aligned} \tag{5}$$

Equation (5) gives an optimization problem. The objective function is non-linear. Moreover, the variables to be determined, which are the r_is, are exponents in the objective function. This kind of optimization problem is hard to solve[2]. We propose CaB, a greedy algorithm, which is presented as Algorithm 1.

Algorithm 1 CaB, algorithm to compute number of replicas for each file class

Input: $S, T, \bar{f}, m, W_i, N_i, B_{ij}$

1: **while** $S > 0$ **do**
2: $\quad i \leftarrow \underset{1 \le i \le m, r_i < T}{argmax} \left(\frac{-W_i \bar{f}^{r_i} \ln \bar{f} \sum_{j=1}^{N_i} (1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C_{B_{ij}}^k (\bar{f})^k)}{\sum_{j=1}^{N_i} B_{ij} \sum_{i=1}^{m} W_i N_i}\right)$
3: $\quad$ **if** $\sum_{j=1}^{N_i} B_{ij} < S$ **then**
4: $\quad\quad r_j \leftarrow r_j + 1$
5: $\quad\quad S \leftarrow S - \sum_{j=1}^{N_i} B_{ij}$
6: $\quad$ **else**
7: $\quad\quad rspace_j \leftarrow S$
8: $\quad\quad S \leftarrow 0$
9: $\quad$ **end if**
10: **end while**

In Algorithm 1, we keep iterating over all file classes and looking for the file class FC_i which has the largest first order partial derivative of the *overall availability* with respect to the current number of replicas assigned to FC_i per file block. We increase the replica number of that file class by 1 and start another iteration, until we use up all storage space. $rspace_i$ stands for remainder space, which is the remaining storage space assigned to FC_i when FC_i is the file class whose replica factor should be increased, but the current available storage space is not enough to store one replica for all files in FC_i. After $rspace_i$ is assigned, it will be utilized to store replicas for smaller files first in FC_i.

When taking the partial derivative of the *overall availability* with respect to the number of replicas of files in FC_i, we get:

$$\begin{aligned} &\frac{\partial \frac{\sum_{i=1}^{m} W_i N_i A_i}{\sum_{i=1}^{m} W_i N_i}}{\partial r_i} \\ &= \frac{\partial \frac{\sum_{i=1}^{m} W_i \sum_{j=1}^{N_i} (1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C_{B_{ij}}^k (\bar{f})^{k r_i})}{\sum_{i=1}^{m} W_i N_i}}{\partial r_i} \\ &= \frac{-W_i \bar{f}^{r_i} \ln \bar{f} \sum_{j=1}^{N_i} (1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C_{B_{ij}}^k (\bar{f})^k)}{\sum_{i=1}^{m} W_i N_i} \end{aligned} \tag{6}$$

When $0 < \bar{f} < 1$, the first order partial derivative of the *overall availability* with respect to the number of replicas of files in class FC_i is positive. The intuition to this result is: when we make more copies of a file and store them at different storage nodes, the chance at least one copy being still available at t_1 increases and the overall availability is improved. Now let us take a look at the second order partial derivative:

[2]Using the weighted geometric mean instead of the arithmetic mean would enable us to transform the objective function to a linear function, but it doesn't capture the fact that each file class should contribute to the overall availability differently based on their importance.

$$\begin{aligned} &\frac{\partial^2 \frac{\sum_{i=1}^{m} W_i N_i A_i}{\sum_{i=1}^{m} W_i N_i}}{\partial r_i^2} \\ &= \frac{\partial^2 \frac{\sum_{i=1}^{m} W_i \sum_{j=1}^{N_i} (1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C_{B_{ij}}^k (\bar{f})^{k r_i})}{\sum_{i=1}^{m} W_i N_i}}{\partial r_i^2} \\ &= \frac{-W_i \bar{f}^{r_i} (\ln \bar{f})^2 \sum_{j=1}^{N_i} (1 - \sum_{k=1}^{B_{ij}} (-1)^{k+1} C_{B_{ij}}^k (\bar{f})^k)}{\sum_{i=1}^{m} W_i N_i} \end{aligned} \tag{7}$$

When $0 < \bar{f} < 1$, the second order partial derivative is negative. This result shows that, as we make more replicas of a single file, the improvement to overall availability per replica decreases. In Algorithm 1, we always choose the file class which will improve the overall availability the most when we have storage space to store replicas, and avoid making non-storage-efficient replicas for a file class. Notice that at step 2 of Algorithm 1, we divide the first order derivative by $\sum_{j=1}^{N_i} B_{ij}$, which is the total number of file blocks in file class FC_i. The purpose is to assign storage space to a file class which can improve the *overall availability* the most *per file block*. The complexity of the CaB algorithm is $\mathcal{O}(m \frac{S}{\min N_i})$.

IV. Evaluation

We compare the CaB algorithm with a baseline algorithm which naively assign storage space proportional to the weight of files. We use the availability metric introduced in § III-A to compare the overall availability. We also compare the availability of each individual file class.

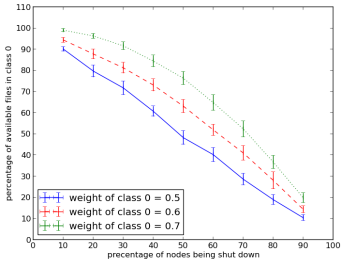
Our evaluation is simulation-based, using the following cluster configuration: 4096 nodes, each with a storage capacity of 32768 blocks. The maximum tolerable probability that a storage node fails during $[t_0, t_1$, $\bar{f}$, is set to be 0.9; note that this as configuration parameter of CaB, and not the actual percentage of nodes failed. During the simulation, we make a fixed percentage of random nodes to fail; this is the actual percentage of failed nodes (as opposed to $\bar{f}$). We vary the percentage of failed nodes from 10% to 90%, in 10% increments. For each percentage of failed nodes (x axis in graphs), we run the simulation and measure the availabilities of files and show the results averaged across 100 runs for each data point. The error bars in the following graphs represent one standard deviation of the availabilities measured in the 100 runs. We perform the experiments a 3.1GHz quad-core machine with 8GB memory.

A. Weight Sensitivity

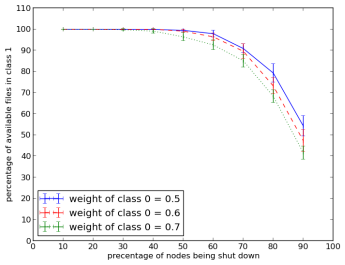
In this section, we change the file class weight for a particular class and observe how it affects availabilities of this file class and all file classes as a whole when we use CaB to assign storage space. As presented in Table II, the input file classes include a lower priority file class which has more files and a higher priority file class which has less files. We vary the file class weight of the lower priority file class from 0.5 to 0.7, and show how it changes the availabilities of two file classes and the overall availability. We set the total available

TABLE II
THE FILE SIZES AND WEIGHTS IN INPUT FILE CLASSES FOR SECTION IV-A

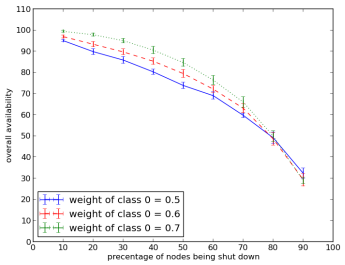
File Class Index	File Class Weight	File Sizes
file class 0	0.5/0.6/0.7	200 1-block files
file class 1	1	100 1-block files



(a) availability of files in file class 0 when weight of file class 0 increases



(b) availability of files in file class 1 when weight of file class 0 increases



(c) overall availability when weight of file class 0 increases

Fig. 1. Impacts of increasing weight of lower priority file class on class availabilities and overall availabilities

storage space to be 900 file blocks and $\bar{f} = 0.9$. The results are shown in Figure 1.

When the weight of file class 0 (the lower priority file class) increases, the direct impact is increased availability for files in file class 0. Furthermore, as we can see in Figure 1a, the availability for files in file class 0 increases in a stable way. The improvements of availability of file class 0 after we increase file class weight from 0.5 to 0.6 and from 0.6 to 0.7 are roughly equal. On the other hand, as a trade-off the availability of file class 1(the higher priority file class) decreases as we increase the file class weight for lower priority file class, yet the impact is smaller. Finally, as shown in Figure 1c, when we increase

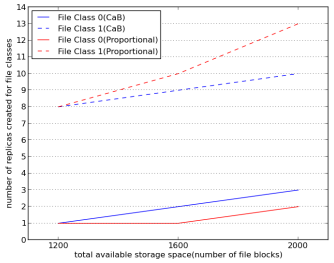


Fig. 2. Number of replicas created by two space assignment algorithms under varied available storage space

the weight of file class 0, the overall availability increases when less than 80% of storage nodes fail sometime during the simulation and decreases when failures are very common (more than 80% of storage nodes fail).

B. Storage Efficiency

In this section, we compare CaB with the baseline algorithm on availability when we increase the amount of total storage space. During the experiment we set the total storage space to be 900, 1200, and 1500 file blocks, which are 3 times, 4 times and 5 times of total number of file blocks in input file classes correspondingly. The file sizes and weights in input file classes are shown in Table IV. The comparison results are presented in Figure 3.

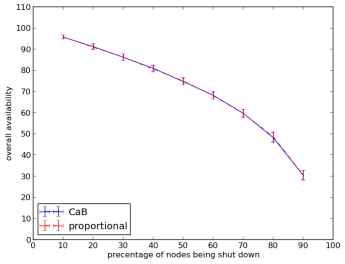
The results show that when the total available storage space is 3 times of the total number of file blocks, the weight-proportional space assignment algorithm achieves the same overall availability as CaB does. However, if we keep increasing the total storage space, CaB starts to outperform weight-proportional algorithm. The reason behind is, as mentioned in Section III, CaB stops creating non-storage-efficient replicas for a file class when there are enough number of replicas of that class created. This result suggests that CaB has better utilization of additional storage space when storage space scales large. We show the number of replicas created by both space assignment algorithms under varied available storage space in Figure 2.

TABLE III
THE FILE SIZES AND WEIGHTS IN INPUT FILE CLASSES FOR SECTION IV-B

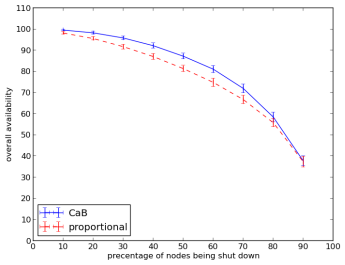
File Class Index	File Class Weight	File Sizes
file class 0	0.5	300 1-block files
file class 1	1	100 1-block files

C. Computation Time

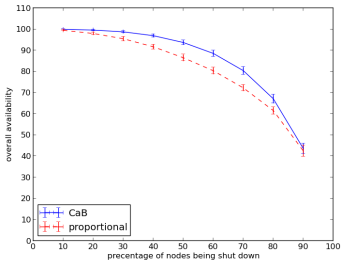
Since we need to run our algorithm periodically to accommodate new arriving data, it is essential that our replica number computation algorithm takes short time to run. In this section, we demonstrate our CaB algorithm can be run fast even when we have relatively high number of file classes and large amount of available storage space compared to file class size.



(a) overall availabilities when $\bar{f}$ = 0.9, total capacity = 900 blocks



(b) overall availabilities when $\bar{f}$ = 0.9, total capacity = 1200 blocks



(c) overall availabilities when $\bar{f}$ = 0.9, total capacity = 1500 blocks

Fig. 3. The comparison of overall availabilities achieved by CaB and weight-proportional space assignment algorithms.

The file class input for this section is presented in Table V. We have ten file classes, and each has an incremental file class weight. Each file class has 100 1-block files. We vary the total available storage space from 5000 file blocks (5 times of total number of file blocks in input file classes) to 50000 file blocks (50 times of total number of file blocks in input file classes), and record the time needed to run the CaB algorithm. The results are shown in Figure 4.

AS we can see from Figure 4, the computation time required by CaB increases almost linearly as the total storage space. Even for a relatively high number of file classes (10) and an amount of total available storage space as large as 50 times of the original input file class size, the computation time required is below 250 milliseconds. This result suggests CaB can be used as an off-line replica number computation algorithm which runs periodically without high computation overhead.

TABLE IV
THE FILE SIZES AND WEIGHTS IN INPUT FILE CLASSES FOR SECTION IV-C

File Class Index	File Class Weight	File Sizes
file class 0	0.1	100 1-block files
file class 1	0.2	100 1-block files
file class 2	0.3	100 1-block files
file class 3	0.4	100 1-block files
file class 4	0.5	100 1-block files
file class 5	0.6	100 1-block files
file class 6	0.7	100 1-block files
file class 7	0.8	100 1-block files
file class 8	0.9	100 1-block files
file class 9	1	100 1-block files

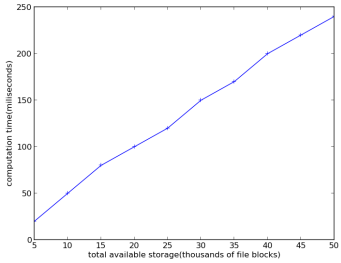


Fig. 4. The computation time required for different total storage space

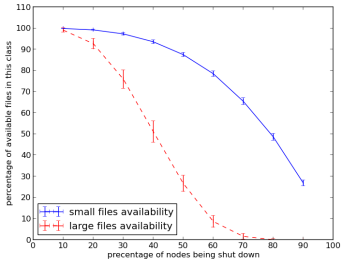
D. Protecting Large Files

In this section we show an application of CaB to protect the availability of larger files. Usually larger files are penalized when all files are assigned the same replication factor in a storage system, because larger files have more file blocks and thus will be more likely to encounter a block failure. For this experiment we have two file classes, where one file class consists of larger files which have sizes of 10 blocks and one file class consists of smaller files which have sizes of 1 blocks, as shown in Table V. We set the total available storage space to be 600 file blocks and $\bar{f} = 0.9$. We compare the performances of CaB and a uniform replica assignment policy which sets the replica factor to 3 for all files. The results are shown in Figure 5.

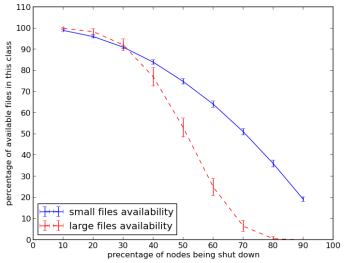
TABLE V
THE FILE SIZES AND WEIGHTS IN INPUT FILE CLASSES FOR SECTION IV-D

File Class Index	File Class Weight	File Sizes
file class 0	0.09	100 1-block files
file class 1	1	10 10-block files

From Figure 5a we can see, a default replica number assignment policy which equally assigns replica number to each file can cause the larger files suffer low availability. After we use CaB and assign a higher weight to large files, though the results show the larger files still have lower availability compared to small files, the difference of availabilities between two file classes has been reduced. Furthermore, for a conservative failure of nodes between $[t_0, t_1]$ (0.20 in x-axis), the large size files can even achieve a higher availability than than small size files.



(a) availabilities of small and large files when using default replica factor = 3



(b) availabilities of small and large files when using CaB

Fig. 5. Comparison of file classes of different sizes under CaB and 3 replicas per file

V. Discussion and Future Work

The current algorithm is static and does not take time into account. In practice: (i) files come and go, (ii) class files change in time, and (iii) nodes fail and are repaired through time. A simple way to address this issue is to apply the current algorithm periodically, say every 24 hours. This can be set-up as an automated task performed during low usage hours. For example, the current implementation of HDFS has a *load balancer* service that must be set-up to run periodically so that it re-distributes the replicas across the nodes to eliminate the unbalance that accumulates over time as files are deleted. However, periodic approaches may be inadequate in environments with high file churn [9].

We are working on evaluating the periodic approach described above, and in providing an evaluation across other dimensions: throughput and load balancing. Having extra replicas of some file classes, and less replicas of other file classes, has an impact in these dimensions, specially if those file classes are based on data access patterns.

Additionally, the random and independent failure model used in this paper is a basic starting point, but a better failure model would enable us to do a more realistic evaluation. We are doing a literature review and collecting some real system statistics to incorporate a better failure model in our study.

Files can be assigned to classes (priorities) based on predetermined user requirements. However, in other situations it may be useful to be able to automatically assign files to classes based on data access patterns. A scheme to monitor file usage and assign classes based on access patterns is desirable.

We addressed the problem of replica number calculation, but a related problem is that of replica placement. In HDFS, the default policy is to locate the first replica in the current rack, a second replica in the same rack as the first one, and all other replicas in random racks [2]. This policy provides a balance between minimizing bandwith usage across racks and maximizing availability in the event of correlated failures (e.g., whole-rack failure). More advanced replica placement schemes, such as Ceph's CRUSH [10], have been proposed to target different performance and reliability goals.

Finally, the problem of keeping replicas consistent is orthogonal to the one studied in this paper. Prior algorithms—like primary-copy, chained or splay replication—can be used for this purpose [4].

VI. Conclusions

We considered the case of using file replication to improve the availability of different classes of files, where some classes are more "important" than others and more replicas are created for them to achieve improved availability. We presented an overall availability metric and an algorithm to compute data replication numbers for file classes with different priorities in a distributed storage system, while utilizing storage space efficiently. Simulation results show that our algorithm is able to improve the availability of the high priority files and has a low computation time, making it suitable for periodic recalculation of replica number assignments.

Acknowledgement

This paper is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proc. ACM Symp. Operating Systems Principles (SOSP)*, 2003.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE Symp. Mass Storage Systems and Technologies (MSST)*, 2010.

[3] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. Symp. Operating Systems Design and Implementation (OSDI)*, 2006.

[4] S. Weil, A. Leung, S. Brandt, and C. Maltzahn, "RADOS: A scalable, reliable storage service for petabyte-scale storage clusters," in *Proc. Intl. Work. Petascale Data Storage (PDSW)*, 2007.

[5] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Disk-locality in datacenter computing considered irrelevant," in *Proc. USENIX Conf. Hot Topics in Oper. Sys. (HotOs)*, 2011.

[6] M. e. a. Abd-El-Malek, "Ursa Minor: Versatile cluster-based storage," in *Proc. USENIX Conf. File and Storage Technologies (FAST)*, 2005.

[7] C. Abad, Y. Lu, and R. Campbell, "DARE: Adaptive data replication for efficient cluster scheduling," in *Proc. IEEE Intl. Conf. Cluster Comp. (CLUSTER)*, 2011.

[8] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Proc. IEEE Intl. Conf. Cluster Comp. (CLUSTER)*, 2010.

[9] C. Abad, N. Roberts, Y. Lu, and R. Campbell, "A storage-centric analysis of MapReduce workloads: File popularity, temporal locality and arrival patterns," in *Proc. IEEE Intl. Symp. Workload Characterization (IISWC)*, 2012.

[10] S. Weil, S. Brandt, E. Miller, and C. Maltzahn, "CRUSH: Controlled, scalable, decentralized placement of replicated data," in *Proc. ACM/IEEE Conf. Supercomp. (SC)*, 2006.