Maintaining Data Reliability without Availability in P2P Storage Systems *

Lluis Pamies-Juarez
Universitat Rovira i Virgili
Tarragona, Spain
lluis.pamies@urv.cat

Pedro Garcia-Lopez
Universitat Rovira i Virgili
Tarragona, Spain
pedro.garcia@urv.cat

ABSTRACT

Peer-to-peer (P2P) storage is a promising technology to provide users with cheap and online persistence. However, due the instability of these infrastructures, P2P storage systems must introduce redundancy in order to guarantee a reliable storage service. Besides, they need data repair algorithms to maintain this redundancy in front of permanent node departures. To ensure that such repairs can always be run, existing P2P storage systems aim to maintain 100% data availability. Unfortunately, this solution seems to overkill in preventing data loses, introducing network and data overheads.

In this paper we propose a new data repair algorithm able to guarantee a high reliable storage service without 100% data availability. The main idea is to ensure that objects are kept stored instead of maintaining them available. We analytically prove that our approach reduces considerably the total amount of redundancy. Moreover, through simulation, we show how our approach significantly reduces the required number of repairs, decreasing both, the network and the storage overheads.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications, distributed databases*; H.2.4 [**Information Storage and Retrieval**]: Information Storage

General Terms

Algorithms, Reliability and Theory

Keywords

peer-to-peer, distributed storage, proactive repair, data availability

*This work has been partially funded by the Spanish Ministry of Science and Innovation through project P2PGRID, TIN2007-68050-C03-03.

SAC'10 March 22-26, 2010, Sierre, Switzerland.

1. INTRODUCTION

Peer-to-peer (P2P) storage systems have received a lot of attention in recent years. This interest arises from the idea of aggregating idle disk resources from desktop computers to build decentralized and collaborative storage systems. Such systems encourage users to share local disk resources to obtain online storage capacity. However, unlike other online storage technologies (i.e. large data centers or clusters of commodity computers), P2P systems are composed of highly unstable nodes affected by churn. To cope with this churn, P2P storage systems need to detect and deal with transient and permanent node disconnections. On the one hand, to cope with transient node disconnections, P2P storage systems seek to maintain high data availability to ensure that data objects can be always be recovered –even when some nodes are offline. On the other hand, permanent node disconnections cause permanent data loses, and hence, an availability drop off. It forces P2P storage systems to regenerate the blocks lost due these permanent disconnections.

In order to treat with both types of churn, P2P storage systems should face two main problems:

1. Decide the amount of data redundancy and the number of nodes required to store each data object.
2. Design data repair algorithms able to reintroduce this redundancy when nodes leave the system.

Several works [3, 2, 15] have proposed solutions for the first problem. These solutions use redundancy schemes based on replication or coding techniques to determine the optimal amount of redundancy. Since an increase in the data redundancy entails an increase in the storage and network overheads, minimizing this redundancy is the main concern of such solutions. In this work we will assume the use of a rate-less erasure code [8] to deal with this first problem, focusing only on the second problem. From the existing data redundancy schemes, rate-less erasure codes are a very interesting solution because they can generate as many redundant blocks as they want in a nondeterministic way, simplifying the overall storage system.

Other works [4, 13, 6] faced the second problem proposing data repair algorithms able to reintroduce the lost redundancy. Usually, these algorithms work by generating new redundant blocks and storing them again in the system. However, because of the constraints imposed by the redundancy scheme, repair algorithms should completely recover each original object before repairing any of its blocks. Due to this, data reliability relies on whether repair process can recover these objects, and existing P2P storage systems

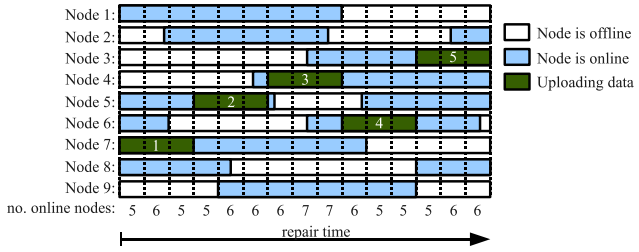

Figure 1: A scenario representing a data repair process. Each node stores 1 redundant block and the repair process spends 3 time units to download each.

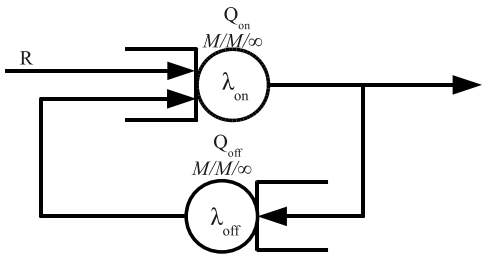

Figure 2: Network of queues used by Duminuco et al.[6] to model the number of online nodes and the required repair rate.

maintain high data availability –close to 100%– for this reason. Unfortunately, maintaining this high availability requires the use of large amounts of redundancy, increasing the storage overhead.

We have observed that repair processes can be usually executed with low data availabilities. To show it, we consider the example depicted in Figure 1. The example shows a system where objects can be recovered from 5 redundant blocks. To maintain 100% data availability the system should ensure that there are always 5 nodes online. Due churn's characteristics it implies storing 9 redundant blocks –the amount required to guarantee a minimum of 5 blocks online. In this scenario, a repair downloading the 5 nodes lasts 15 time units. However, the depicted repair could have finished even without the existence of nodes 1,2,8 and 9 reducing considerably the number of redundant blocks in the system, and therefore, its data availability. Although we did not consider parallel downloads, this simple example shows that maintaining high availability is not so important than existing storage systems claimed.

In this paper we propose a new data repair strategy for P2P storage systems able to maintain the same reliability than existing ones, but reducing the required redundancy. The main idea behind our strategy is that storage systems could reduce from 100% data availability, to the minimum that ensures that repair process can always be run. To find this minimum availability, we use the model presented in [6]. Besides the required redundancy, this model allows us to adapt the in from of dynamic node availabilities and to measure how often repairs should be scheduled. We analytically prove that our strategy reduces the storage overheads up to 25% in typical scenarios, using less number of repairs and then, reducing the network burden.

For the rest of the paper, we firstly give in Section 2, a description of the existing mechanisms designed to maintain data reliability and we describe the models and algorithms used as the basis of our work. Section 3 introduces our data repair algorithm. Finally, in Section 4, we present the simulations and the results obtained from our experimentation.

2. BACKGROUND AND RELATED WORK

Redundancy schemes based on data coding techniques are a common solution for P2P storage systems. Unlike fixed-rate codes like Reed-Solomon or Tornado Codes [9] where the maximum number of redundant blocks is initially determined, rate-less erasure codes [8, 10, 12] can generate an unlimited number of redundant blocks. This property allows data repair algorithms to generate new redundant blocks without being aware of the other blocks stored in the system. This property makes this kind of codes very suitable for being used by data repair algorithms. In this paper we assume a storage system using a rate-less erasure code that splits data objects into m redundant blocks. After it, n redundant blocks are generated and each one stored to a different node. Thanks to the redundancy scheme, the storage system ensures that original objects can be always recovered by gathering m out of the n redundant blocks.

The above scheme allows storage systems to generate the redundancy required to guarantee high data availability. However, when nodes abandon the system, redundancy is reduced and this availability could be threatened. Repair algorithms are the responsible to cope with these departures and ensure that the number of redundant blocks is always kept close to n. Traditionally, these repair algorithms used *reactive* schemes [5, 3, 11] that triggered the reparation lost blocks as soon as node disconnections were detected. The main problem of this approach is that it is not aware of temporal node disconnections, considering as lost the blocks stored in temporally unavailable nodes. To solve this problem other *reactive* schemes [1, 7] differentiated between permanent and transient failures, being able to reintegrate data from temporally unavailable nodes and reducing the total number of repairs. However, these reactive schemes suffer from network spikes caused by data repair processes. Due to this problem, systems like [13, 6] went a step further and proposed *proactive* repair algorithms able to smooth these spikes. Unlike reactive algorithms, proactive algorithms schedule the creation of new redundant blocks at a constant rate independently of the actual node departures. By estimating the churn characteristics and setting the repair rate consequently, proactive repair can maintain the number of online nodes close to n avoiding bandwidth spikes.

From the above algorithms, the proposed by Duminuco et al. [6] is, to the best of our knowledge, the most complete work on this area. What make this work so interesting is that it is able to dynamically measure the churn fluctuations and adapt the repair rate accordingly. To do it they used the network of queues represented in Figure 2 that models a P2P storage system. This network consists of two $M/M/\infty$ queues named Q_{on} and Q_{off} where customers waiting on them represents respectively, online and offline nodes. Once served, nodes from Q_{on} are queued in Q_{off} with probability $1-\alpha$ or dropped out of the system with probability α. Nodes served in Q_{off} are queued again in Q_{on}. Finally, the new redundant blocks created by the repair process are modeled

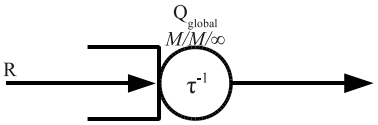

Figure 3: Queue that models the nodes participating in the system.

as new customers entering Q_{on} at a constant rate R. Service rates of both queues are λ_{on} and λ_{off} respectively. It means that $t_{on} = \lambda_{on}^{-1}$ and $t_{off} = \lambda_{off}^{-1}$ are the mean waiting times of each queue. Denoting as L_{on} and L_{off} the probability distribution of the number of customers queued in the online and offline queues, Duminuco et al. pointed out that both distributions can be approximated using a Normal distribution with values:

$$E\left(L_{on}\right) = var^2\left(L_{on}\right) = \frac{R}{\alpha\lambda_{on}} = \frac{\mu R}{\alpha} \quad (1)$$

$$E\left(L_{off}\right) = var^2\left(L_{off}\right) = \frac{(1-\alpha)R}{\alpha\lambda_{off}} = \frac{(1-\alpha)(1-\mu)R}{\alpha} \quad (2)$$

where μ represents node's availability –the proportion of time nodes spent online.

Duminuco et al. use the approximation of the number of online nodes to a Normal distribution to find out the repair rate that ensures a mean number of online nodes equal to n: $R = (\alpha n)/\mu$. Using this expression, the system can use live estimators of α and μ and adapt the repair rate to guarantee that the mean number of online nodes is kept steady even with the presence of churn fluctuations. However, because L_{on} follows a Normal distribution, with infinite tails, there exists a small probability that the number of online nodes falls below m, $\Pr[L_{on} \leq m] > 0$. To avoid it and ensure that data is never lost, Duminuco et al. use a hybrid repair scheme combining the proactive repair rate R with a reactive repair algorithm that trigger additional data repairs when the number of online nodes drop below a predefined threshold T, where $n > T > m$. Using this threshold the systems ensures that the number of online nodes is always greater than it. Using this hybrid scheme they proved that when n and T parameters are properly chosen, 100% data availability can be maintained using very few reactive repairs.

3. DATA RELIABILITY WITHOUT AVAILABILITY

As we have presented in the previous section, proactive data repair processes are a very interesting solution for maintaining data redundancy in P2P storage systems. However, as we shown in the example from Section 1, data reliability do not requires of 100% data availability. Since in P2P storage systems some disconnected nodes reconnect in a short period of time, data repair processes could be designed to consider that the data stored in temporally unavailable nodes would be accessible in a near future. In this section we exploit this idea but leveraging the knowledge acquired by previous data repair algorithms. To do it, we analyze the effects of our new repair strategy in the storage model presented by Duminuco et al. [6].

Utard et al. [14] determined that the mean node lifetime τ (time node spent in the system before abandoning) is exponentially distributed with the following mean value:

$$\tau = t_{on} + \frac{1-\alpha}{\alpha}(t_{on} + t_{off}) \quad (3)$$

They simplified this expression by properly setting the time basis so that $t_{on} + t_{off} = 1$, they obtained that $\mu = t_{on}$ and then,

$$\tau = \mu + \frac{1-\alpha}{\alpha}. \quad (4)$$

Considering this lifetime, we introduce the additional queue model depicted in Figure 3 to determine the number of nodes in the system and the time that they spend on it. Similar to the model represented in Figure 2, our model is formed by a single queue where nodes are queued until they definitively abandon the system. At every instant, this queue contains the nodes waiting on both Q_{on} and Q_{off} queues. Because of Utard's lifetime expression we know that the service rate of this queue is τ^{-1}, so we can use Little's law to obtain the average number of customers is τR, so the number of nodes in the system can also be approximated to a Normal distribution with parameters:

$$E\left(L_{global}\right) = var^2\left(L_{global}\right) = \tau R \quad (5)$$

Unlike Duminuco's repair rate that aims to maintain 100% data availability, so $E(L_{on}) = n$, we propose to set the repair rate to maintain the same amount of redundancy stored in the system. Using our global queue, our problem is to find the repair rate R ensuring that data is kept stored, $E(L_{global}) = n$. We will refer to both strategies as *keep-avail* strategy and *keep-stored* strategy respectively:

- Under the **keep-avail strategy**, the proactive repair rate R_A is set to ensure that the mean number of online nodes is equal to the targeted nodes, $E(L_{on}) = n$. Using Expression (1), they obtained that $R_A = (\alpha n)/\mu$.
- Under the **keep-stored strategy**, the proactive repair rate R_S is set to ensure that the mean number of nodes in the system is equal to the targeted nodes, $E(L_{global}) = n$. Since $E(L_{global}) = E(L_{on} + L_{off})$, using the Cramér's theorem we can obtain the required rate R_S able to maintain this targeted number of nodes in the system, $R_S = (\alpha n)/(\alpha(\mu - 1) + 1)$.

Definition 1. *The availability overhead f is the overhead that a storage systems suffers by using the keep-avail strategy instead of the keep-stored strategy. This overhead is defined as:*

$$f = \frac{R_A}{R_S} = \frac{\alpha(\mu-1)+1}{\mu} \quad (6)$$

From the f's definition we obtain that:

Theorem 1. *The availability overhead of using keep-avail strategy instead of keep-stored strategy is always greater than one: $f \geq 1$.*

Proof. We will prove the theorem by contradiction. Let us assume that the it is false,

$$f < 1$$

$$\frac{R_A}{R_S} < 1$$

since $R_S > 0$,

$$\begin{aligned} R_A &< R_S \\ \frac{\alpha n}{\mu} &< \frac{\alpha n}{\alpha(\mu-1)+1} \\ \mu &> \alpha(\mu-1)+1 \\ \mu - 1 &> \alpha(\mu-1) \end{aligned}$$

since μ is defined as a probability, $\mu \in [0,1]$, so $\mu - 1 \leq 0$, then,

$$\begin{aligned} \alpha &> \frac{\mu-1}{\mu-1} \\ \alpha &> 1 \end{aligned}$$

which is a contradiction because α was defined as a probability too, so $\alpha \in [0,1]$, and the theorem follows. □

From Theorem 1 we directly infer that:

COROLLARY 1. *The proactive repair rate used by the keep-avail strategy is greater or equal than the used by the keep-stored strategy:* $R_A \geq R_S$.

Furthermore, analyzing total number of nodes in the system we obtain following theorem:

THEOREM 2. *The storage overhead in a system using a keep-avail strategy is* f *times greater than using a keep-stored strategy.*

PROOF. The storage overhead is the total amount of redundant blocks stored in the system. Since this value follows a Normal distribution (Eq. 5), the statement of this theorem can be rewritten as $E(L_{globalA}) = f \times E(L_{globalS})$. Where $L_{globalA}$ and $L_{globalS}$ are the probability distributions of the number of nodes in the system using keep-available and keep-stored strategies respectively. Then,

$$\begin{aligned} E\left[len(Q_{global})_A\right] &= fE\left[len(Q_{global})_S\right] \\ \tau R_A &= f\tau R_S \\ f &= \frac{R_A}{R_S} \end{aligned}$$

which is true by Definition 1 and the theorem follows. □

From the above theorem and using Equation 1 we can also infer that:

COROLLARY 2. *The average number of online nodes using keep-avail strategy is* f *times larger than using keeps-stored strategy:* $E(L_{onA}) = f \times E(L_{onS})$

Summing up, we proved that keep-stored strategy can significantly reduce the storage overhead and the number of repairs required to maintain data reliability. However, since the number of online nodes is highly reduced, there are two issues that should be considered:

1. If the same reactive threshold T is used in both strategies, the number of repairs triggered in keep-stored strategy will be significantly increased due the lower number of online nodes. We need to check whether this increase in the number of reactive repairs do not exceed the reduction in the proactive ones.

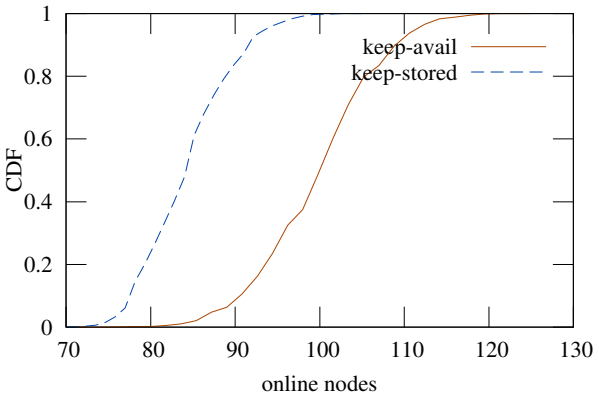

Figure 4: Cumulative distributed function of the number of online nodes in the system.

2. A lower number of online nodes could increase the repair times because of temporal unavailabilities. The storage system must keep low the repair times in order to allow user fast data retrieval and avoid the overlap of consecutive data repairs.

Since repair times follow an unknown distribution and both issues depend on this distribution, we will need to study their effects through experimentation.

4. EVALUATION

We have designed an event based simulator to validate the analytical results presented in previous sections. Our churn parameters are set as in [6], where $\alpha = 0.5$ and t_{on}, t_{off} are set to $\frac{2}{3} \times 24h$ and $\frac{1}{3} \times 24h$ respectively. Using these parameters we obtain that $\mu = 0.66$ and $f = 1.25$. To compare our results with the results presented in [6], we use $n = 100$ as the number of redundant blocks targeted for *keep-stored* strategy, and using Corollary 2, $n = 125$ for the*keep-avail* strategy. Both n values correspond to the mean number of nodes queued in Q_{global}. Finally we set the amount of blocks required by the repair process to $m = 70$ and the reactive threshold set to $T = 75$ for both strategies.

The simulation was run for both strategies during a period of 6 months. In order to simulate bandwidth limitations we assumed that repair processes needed 26 seconds to download each of the 70 blocks. Because of this limitation, the time spent by repairs were approximately 30 minutes for both strategies with a negligible variance. So the repair time in keep-stored has not increased.

During the 6 months period both strategies were able to maintain the object stored. However, keep-stored required less online nodes than keep-avail. Using Corollary 2 we can read this reduction as a redundancy saving for keep-stored strategy. Figure 4 shows the cumulative distribution function of the number of online nodes sampled every R^{-1} seconds. On this figure we can clearly see this reduction in the number of online nodes.

Finally, using the expressions for R_S and R_A from Section 3 we obtain than keep-stored strategy schedules proactive repairs every 24 minutes while keep-avail does every 19 minutes. This means that our strategy requires 21% less proactive repairs. However, since the number of online nodes

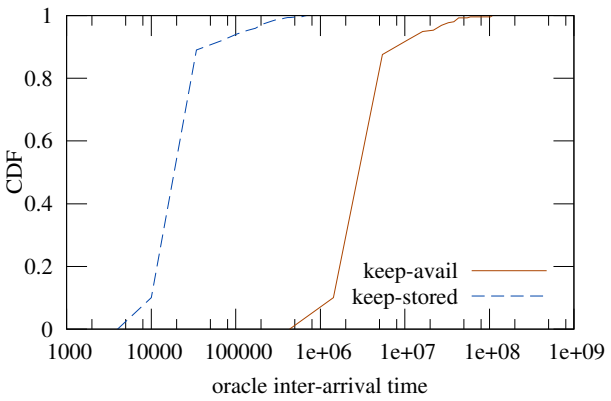

Figure 5: Time between consecutive reactive repairs triggered by the threshold T.

is significantly lower, the number of reactive repairs triggered by T has increased significantly. Figure 5 shows the cumulative distribution of the time between reactive repairs in both strategies. The results show an important increase in the number of reactive repairs but the total number of repairs measured during the entire simulation (reactive ones + proactive ones) is a 15% lower, being 13,656 for keep-avail and 11,647 for keep-stored. This reduction in the number of repairs implies a reduction in the network burden.

5. CONCLUSIONS

In this paper we presented a new data repair strategy for P2P storage systems. Instead of trying to maintain data 100% available, we just ensure that it remains stored in the system, independently of node's availabilities. We analytically proved that our strategy can reduce the required redundancy up to 25% in common scenarios, and we used simulations to show that this reduction is achieved with 15% less data repairs. Although proactive repairs can smooth the network spikes caused by reactive ones, we demonstrated that by reducing the repair rate P2P storage systems are able to maintain the storage service at a lower network and storage costs.

6. FURTHER WORK

We see possible extensions of the analysis presented in this paper in the following directions. First, the presented model considers a storage system with a single data object. However, in real P2P storage systems nodes could store redundant blocks from different data objects. Our objective is to study how repair processes could efficiently repair departed nodes when they were storing hundreds or thousands of redundant blocks. Secondly, we did not allow parallel downloads in repair processes. In further works we will analyze how repair processes could be optimized with parallel downloads and the impact of this optimization in both strategies. And finally, we will study the distribution of the repair times in order to be able to analytically measure the impact of parameters like the size of the redundant blocks or node's bandwidth.

7. REFERENCES

[1] B. G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, and J. Kubiatowicz. Efficient replica maintenance for distributed storage systems. machine availability estimation. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[2] L. Cox and B. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[4] Anwitaman Datta and Karl Aberer. Internet-scale storage systems under churn. a study of the steady-state using markov models. In *Proceedings of the 6th International Conference on Peer-to-Peer Computing (P2P)*, 2006.

[5] P. Druschela and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS)*, 2001.

[6] A. Duminuco, E. W. Biersack, and T. En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *Proceedings of the 3rd CoNEXT conference (CONEXT)*, 2007.

[7] Ranjita Bhagwan Kiran, Kiran Tati, Yu-chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[8] M. Luby. Lt codes. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[9] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the 29th Annual Aymposium on Theory of Computing (STOC)*, 1997.

[10] P. Maymounkov. Online codes. Technical Report TR2002-833, New York University, 2002.

[11] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. *SIGCOMM Comput. Commun. Rev.*, 35(4):73–84, 2005.

[12] A. Shokrollahi. Raptor codes. *IEEE/ACM Transactions on Networking (TON)*, 14, 2006.

[13] E. Sit, A. Haeberlen, F. Dabek, B. Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[14] G. Utard and A. Vernois. Data durability in peer to peer storage systems. In *Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, 2004.

[15] Z. Zhang and Q. Lian. Reperasure: replication protocol using erasure-code in peer-to-peer storage network. In *Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS)*, 2002.