Availability-based methods for distributed storage systems

Anne-Marie Kermarrec, Erwan Le Merrer, Gilles Straub, Alexandre van Kempen

Availability-based methods for distributed storage systems

Anne-Marie Kermarrec
INRIA Bretagne Atlantique

Erwan Le Merrer
Technicolor

Gilles Straub
Technicolor

Alexandre van Kempen
Technicolor

ABSTRACT

Distributed storage systems rely heavily on replication to ensure data availability as well as durability. In networked systems subject to intermittent node unavailability, replicas need to be maintained (i.e. replicated and/or relocated upon failure). Repairs are well-known to be extremely bandwidth-consuming and it has been shown that, without care, they may significantly congest the system. In this paper, we propose an approach to replica management accounting for nodes heterogeneity with respect to availability. We show that by using the availability history of nodes, the performance of two important faces of distributed storage (replica placement and repair) can be significantly improved. Replica placement is achieved based on complementary nodes with respect to nodes availability, improving the overall data availability. Repairs can be scheduled thanks to an adaptive per-node timeout according to node availability, so as to decrease the number of repairs while reaching comparable availability. We propose practical heuristics for those two issues. We evaluate our approach through extensive simulations based on real and well-known availability traces. Results clearly show the benefits of our approach with regards to the critical trade-off between data availability, load-balancing and bandwidth consumption.

1. INTRODUCTION

Digital data is becoming increasingly common (photos, videos, documents) and increasingly important as the digital revolution expands all over the world. As storage needs grow with data production, capacities of modern disks are thus compelled to follow the same tendency. Several studies [16, 10, 2] show that end user hard drives are on average half empty and represent a considerable amount of unused storage space available at the edge of the network. Indeed the authors in [2] reveal, after a five-year study, that the ratio of free space over the total file system capacity remained constant over the years. Therefore we can reasonably argue that the available free space at the edge of the network will follow the same trend as the user data increase. Aggregating this free space, represents therefore a real and scalable *Eldorado* of available space. Distributed storage is a typical example of applications where this idle storage space could be leveraged. In such a distributed storage system, each node is in charge of storing the data of other nodes in exchange for having its own data stored and made available in a durable way. However performances largely depend on the nodes availability. More specifically, nodes may join and leave the system at will without previous warning.

Furthermore designing a *pure* peer to peer architecture in order to offer a reliable distributed storage system may be impractical due to high churn as well as the fact that peers have no incentives to store the data of others [5]. To solve these issues while moving towards practical system deployment, hybrid architectures have been very recently proposed [31, 12]. They add some centralization for reliability or efficiency, and impose incentive mechanisms [31, 21] to discourage selfish nodes. However, even relying on a hybrid architecture, managing the volatile part of nodes remains a complex task. In fact when a node leaves the system unexpectedly, the data it is responsible for is also unavailable. To face this volatility, distributed storage systems replicate data on more than one node so as to tolerate multiple node departures and/or failures. Typically a clever initial replica placement may either for the same replication degree improves data availability or for a given availability may reduce the number of replicas. Moreover, once a degree of replication is set, it has to be kept steady despite nodes leaving the system, in order to guarantee the durability of data. This maintenance mainly involves two operations: *(i)* node failure detection; *(ii)* replica repairs. In this paper, we focus on the detection mechanism, which may waste bandwidth unnecessary if it is inaccurate. Interestingly enough, some well-known works [5, 29] reveal that one of the key aspects of an efficient distributed storage mechanism is precisely the bandwidth consumption.

In practical systems, distinguishing a permanent failure from a transient one with a high success rate is challenging (this is theoretically impossible in asynchronous systems). Yet, significant bandwidth could be saved by triggering repairs only in the event of a permanent failure, assuming that the data is only temporarily inaccessible in the case of a transient unavailability. Therefore too aggressive a repair mechanism may congest the system because of an excessive and useless bandwidth utilization. On the contrary, a low reactivity in

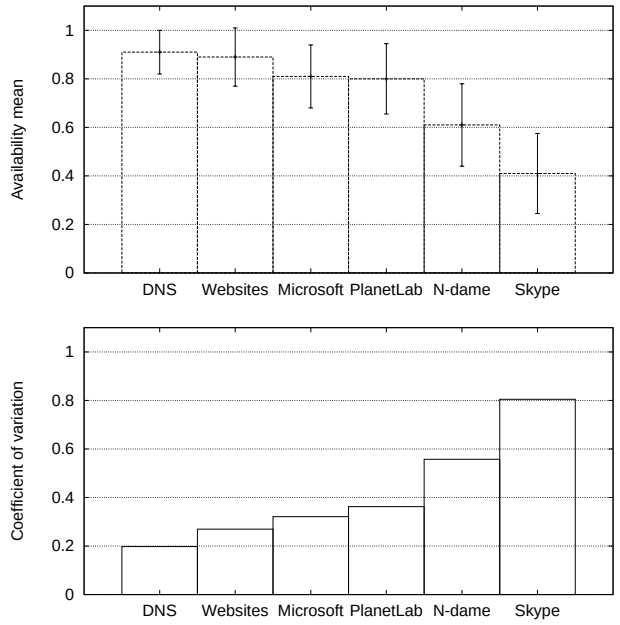

Figure 1: Availability mean and dispersion for various existing systems

the repair process may result in degradation of quality of service (*i.e.* data availability and durability).

In this paper we take up the challenge of tackling this trade-off while answering significant questions closely related to the design of an efficient distributed storage system, namely: *(i) Where should the replicas be placed?*, *(ii) When should the system conclude on a transient or a permanent failure thus triggering a repair?*
We argue that a one-size-fits-all approach is not sufficient and that the statistical knowledge of *every single node* availability, as opposed to system-wide parameters, provides a means to tackle those questions efficiently. In other words, while most previous works have either assumed that nodes are homogeneous or that simple averages on behaviors are representative, we account for the heterogeneity of nodes availability, in order to decide where replicas should be placed as well as when repairs should be triggered.

Contributions. In this paper we propose two availability-based methods to decide *(i)* where to place replicas and *(ii)* when to trigger a repair using a limited availability history of storage nodes. We make the following contributions:

Our replica placement algorithm relies on discovering complementary availability patterns (that we name *anti-correlated*) so as to maximize the availability of data across time. Through extensive simulations based on real availability traces we show that this approach outperforms traditional random placement, resulting in a significantly lower number of replicas for a given availability while load balancing is maintained. This includes facilities for *erasure codes* [34]. For instance, on an availability trace of the Skype network, 5 replicas are needed instead of 8 with random placement, for an equivalent availability of around 99%.

Our repair mechanism trades traditional system-level timeout based on network-wide statistics against a per node timeout, based on the node availability patterns. Nodes self-organize to compute their adapted timeout in a probabilistic way, according to their own behavior and to the current replication factor. Experimental results show that we achieve a lower number of repairs, while preserving the same level of availability of data.

The combination of these methods sets the scene for efficient placement and repair strategies in large-scale distributed storage systems, improving the overall performance on realistic replication scenarios. For instance, on the same Skype trace, our method yields a 37% saving in repairs, while offering the same availability of data.

The rest of the paper is organized as follows: first we present our main motivations for this work in Section 2. Section 3 introduces the system model we consider. Our placement and repair strategies are then detailed in Sections 4 and 5 respectively. In Section 6, we jointly use these two techniques and discuss some practical issues. Then we present related work in Section 7 before concluding the paper in Section 8.

2. LEVERAGING PREDICTABILITY AND HETEROGENEITY

In this section, we review the three main motivations of our availability-aware methods, namely: *(i)* leveraging node predictability with respect to availability, *(ii)* accounting for heterogeneous node availability patterns, and *(iii)* improving the trade-off between replication rate, availability and bandwidth. For instance, these factors can enrich performance-oriented storage systems, that are currently availability-agnostic with regards to the set of edge nodes they are using as a storage basis [12].

2.1 Availability and Predictability

In the context of distributed storage systems, where nodes hosting data may leave and join at will, many systems that do not explicitly deal with specific availabilities of nodes were originally proposed [27, 8]. In the quest for better performance, some works have studied and identified trends in the availability of the hosting resources [9, 3, 18, 6]. The important question of leveraging predictability has then been addressed [22, 18] to handle transient failures efficiently.

In this paper we propose to leverage this observed predictability in the behavior of nodes to provide adapted heuristics that overcome availability-agnostic replica placement and repairs.

2.2 Heterogeneous availability patterns

Most of the systems that account for availability of resources rely on a single system-wide parameter, typically the mean or the distribution of availabilities of all nodes of the system [35, 30, 4, 31]. While it is convenient to apply theoretical models such as Markov models or basic probabilities for the number of replicas to create, recent comments un-

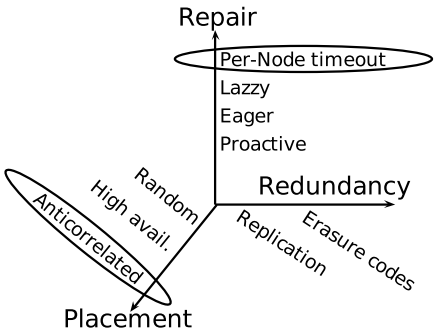

Figure 2: Some design parameter space of a storage system (the approaches proposed in this paper are circled)

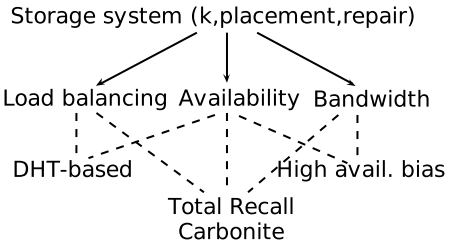

Figure 3: A classification of distributed storage systems

derline why such models have very limited applicability in practice [13, 11].

While some storage systems use platforms such as home gateways [32] that have a homogeneous and high availability, the majority of deployment platforms exhibit a non-negligible heterogeneity in practice. To illustrate our claim, we plot on Figure 1 the mean, the standard deviation, and the coefficient of variation of availabilities of nodes composing systems such as Microsoft desktop computers, Skype or PlanetLab[1]. The figure shows a significant variance in node behavior. This is confirmed in a recent storage system analysis [31]. This trend is even more striking for the two leftmost systems. This demonstrates that availability cannot be accurately expressed by a basic scalar mean trying to represent the overall trend of the fraction of time nodes are up. Furthermore, reducing availability to a mean or a distribution [24, 31] ignores information about node availability patterns while such information could be leveraged to increase the reliability of distributed storage systems [22, 28, 18]. This calls for a finer grain study accounting for specific node availability patterns.

2.3 Storage System trade-offs

Blake et al. [5] have shown that for dynamic storage systems using redundancy, a severe bandwidth bottleneck appears, when maintaining a high data availability. In order to propose realistic system designs, recent works have thus relaxed the 6 nines constraint on data availability, taking more pragmatic replication rates into account.

The third motivation for our work is precisely the critical trade-off between the storage overhead, the load balancing and the bandwidth consumed by the system. A distributed storage system targeting data availability may be represented by a set of three strategies (Figure 2) : *(i)* a redundancy strategy characterized by k (k may be either a replication factor or the rate of an erasure code), *(ii)* a placement strategy (where to replicate the data) and *(iii)* a repair strategy (when and how to repair a lost replica). Those three strategies characterize the trade-off between load-balancing, availability and bandwidth, allowing a classification of distributed storage systems, is presented in Figure 3.

In this formalism, "basic" DHT-based systems such as [8, 27] reach high availability due to very high replication rates (for instance PAST stores a chunk of replica on its whole *leafset* in the DHT), while load balancing is ensured by a pseudo random placement strategy (the hash function balances the load evenly). Furthermore, as pointed out in [19], replicas have to be relocated each time a node is inserted in, or leaves, the replication neighborhood of a file, triggering cumbersome maintenance operations.

Solutions that are availability-aware suggest biasing the placement of data towards highly available nodes [22, 25], thus minimizing bandwidth due to repair cost caused by transient nodes. Nevertheless, this creates a high pressure on stable nodes that are required to contribute significantly more than average nodes. Likewise, for a given availability, reducing the number of replicas decreases the probability a repair is needed. In turn, this has a direct impact on the repair overhead. A clever replica placement algorithm can precisely limit the number of replicas required to ensure an availability degree.

Relaxing one of these three constraints can limit the complexity of a proposed system, yet causes a hardly usable solution in practice. Finally, very few storage systems such as Total Recall [4] or Carbonite [7] address this trade-off by using random placement and non-trivial repairs in order to maximize availability. This is typically what we address in this paper.

One of our motivation is precisely to improve the latter and more challenging class of systems, by proposing availability-aware methods that are applicable as "plugins" to their architecture. Obviously these plugins may also be implemented on the volatile part of hybrid architectures such as those proposed in [31, 12].

3. STORAGE SYSTEM MODEL

Storage systems that leverage available disk space at the edge of the network are using nodes that range from user computers [15] to hardware boxes such as Internet provider gateways [32]. These systems are either fully distributed [17, 27, 19, 4, 7] or hybrid architectures [31, 12]; our availability-aware solution is specifically designed to be applied to any type of network logic where nodes may exhibit

[1]Those availability traces, from scientific publications, are made available in a repository [1].

temporary and possibly recurrent, periods of unavailability. Systems with a near-perfect availability of their components obviously have no need for such a study. Furthermore, even systems exhibiting unpredictability of some significant part of nodes of the network can leverage our approach.

For the sake of clarity and because we focus on the replica and repair strategies, we assume the existence of a service providing requesting nodes with a set of accurate partners. These partners are then used by the requesting nodes to place replicas or repair replicas; replication *clusters* are then created for each data to be stored, following for instance the replication in DHTs leafsets where groups monitoring and storing the same data are created [27, 8]. While such a service is trivial to implement using a server, it is directly applicable to pure distributed systems, where nodes can collaborate to achieve this computation, through *gossip* for example [18, 28]; each node in a cluster then monitors other cluster nodes to detect failures.

In order to exploit information on node availability, our system must keep track of a limited history of those availability and unavailability periods. In practice, such availability history can be maintained by one or a few servers, the node itself providing it on demand, or by a distributed monitoring system if nodes cannot be trusted [23]. This availability history is represented as a vector of a predefined size (acting as a sliding window of time). For each time unit, the corresponding vector entry is set to 1 if the particular node was online at that time and -1 otherwise. In this paper, we assume one-week history vectors and one-hour time unit. This length has been shown to capture most user behaviors accurately (*e.g.* diurnal and week end presence patterns) [3, 22, 18, 9].

Since we focus on data availability, we do not consider deviating node behavior (*e.g.* free riders).

Finally, as our aim is a pragmatic study of what can be achieved beside purely theoretical models for placement or timeouts, we use as a basis publicly available traces, that have been deeply studied in their respective original papers [1]. Those traces come from a wide range of systems; when applying techniques on them, the goal is to underline tendencies for the associated kind of availability they exhibit, more than just proposing a specific improvement for a narrow range of systems.

4. AVAILABILITY-AWARE PLACEMENT STRATEGY

In this section, we propose an answer to the question: *Where should replicas be placed so as to maximize availability while ensuring an evenly balanced load ?* The availability-aware placement strategy proposed relies on leveraging the monitored availability of nodes.

4.1 The R&A placement strategy

Our goal, as presented in Section 2, is to offer a better data availability than a random placement strategy using the same storage overhead k and without getting a high skewed load distribution. Thus, this excludes biased placement towards highly available nodes, which by definition only considers specific nodes for the whole system satisfaction. Instead, we leverage the knowledge of the availability of each node in order to choose other nodes to place the replicas. Those nodes are chosen so that their availability patterns match the unavailability periods of the requesting node. Such nodes are denoted as *anti-correlated* nodes as we explain below.

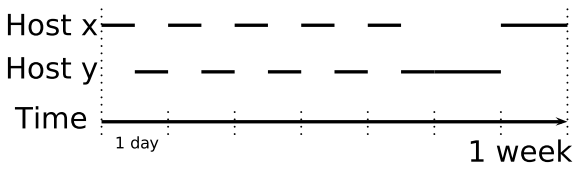

Figure 4: Perfect anti-correlation between nodes x and y.

We define the notion of availability anti-correlation, for a requesting node x as the opposed presence of a node y on the same predefined period (thus minimizing overlapping periods of uptime). Figure 4 illustrates this notion: node y is perfectly anti-correlated to node x, as it is online during all periods of unavailability of x. Comparing vectors of node availability history, nodes can be sorted by their effective correlation to a given node x.

Let $\overrightarrow{A_x}$ and $\overrightarrow{A_y}$ be the availability vector of respectively node x and y. In practice, the (anti)correlation is modelled as the angle, noted $\Theta_{x,y}$, between the two vectors $\overrightarrow{A_x}$ and $\overrightarrow{A_y}$.

$$\Theta_{x,y} = \arccos\left(\frac{\overrightarrow{A_x} \cdot \overrightarrow{A_y}}{\|\overrightarrow{A_x}\| \cdot \|\overrightarrow{A_y}\|}\right)$$

- $\Theta_{x,y} = 0$: Perfect correlation between nodes x & y
- $\Theta_{x,y} = \pi/2$: No correlation between x & y
- $\Theta_{x,y} = \pi$: Perfect anti-correlation between x & y

Our placement strategy relies on building clusters (i.e. sets of nodes holding replicas of the same data) with *pairs* of nodes exhibiting anti-correlated behavior so as to cover the whole prediction period. Since our goal is to increase the global availability, an anti-correlated node to a *reference node* is the best candidate to patch the time when this *reference node* is offline, as illustrated in Figure 4. By picking a node, finding its best anti-correlated counterpart and iterating until the cluster contains enough nodes to replicate data k times , the effect of time patching is increased. We call this scheme R&A, for Random and Anti-correlated placement scheme.

In order to overcome a random strategy (where the cluster is built up with nodes chosen uniformly at random) using our placement method, part of the system nodes must have some predictability on their availability behavior. The more predictable the nodes in the system are, the more efficient is our method when compared to availability-agnostic placement. On the contrary if all nodes in the system show a random behavior then the R&A scheme is equivalent to a

random one, as our placement also leverages random selections. Recent works on availability prediction of at least a subset of the network back up this claim [22, 18, 28].

4.1.1 R&A method core

When a new data item has to be stored, a new cluster is created, where each node belonging to this cluster stores a replica (nodes may of course participate in more than one cluster). This cluster is filled as follows: first the system randomly selects a *reference node*; this selection might be achieved using a *random walk* in a decentralized system for example or simply by a uniform sampling in a hybrid system. Then the node whose behavior is the most anti-correlated to this reference node is selected to form a pair of anti-correlated nodes. This pair is then added to the cluster. This process is iterated until the number of nodes in the cluster is equal to the system replication factor k. In case of an odd k, an additional random node is included in the cluster. The randomness inherent to the selection of devices in the cluster is intended to ensure load balancing while pairs of anti-correlated behaviors improve data availability.

4.1.2 Experimental evaluation of R&A

In order to evaluate our R&A placement strategy we use a public trace of Skype [14], which exhibits a high heterogeneity among availabilities of nodes. As *availability of data* is defined as the presence of at least one replica at anytime, random placement, as well as R&A, performs well on stable traces such as DNS, Microsoft or Planet Lab. In fact a majority of the nodes are always up (Figure 1); this ensures with high probability that if they are selected, the replicas they host are sufficient to achieve the targeted availability. Deploying an advanced placement strategy is thus of no interest on this type of trace. In the Skype trace, nodes having an uptime below 1% have been removed from the trace, resulting in a number of alive nodes equal to 1901.

We conducted the simulations as follows: we assume a homogeneous storage demand, then each of the *alive* nodes seeks to back-up one data item, with different replication factor k (from 2 to 10). For instance, for $k = 2$ the number of objects to be stored in the system is thus 3802. We consider a two-week period for the evaluation: the anti-correlation between availability behaviors is computed over the first week (training period). The evaluation of data availability is performed on the second week. Data is considered available if at least one node holding a replica is online for *each time unit* of the entire evaluation period. Then we evaluate the availability mean compared to both a random placement strategy and a strategy biased towards highly available nodes . Availability mean and standard deviation are plotted for different replication factors k. Results are depicted in Figure 5.

The strategy biased towards highly available nodes (called highly available strategy for brief) is included in the evaluation in order to assess its impact on availability and even more on load balancing. This strategy, mainly simulated to back-up our claim about the high skew produced on load balancing, is straightforward and has already been used as a point of comparison in [22]. For each replica to store, 10 nodes are first randomly selected. Then, within this 10 node subset, the one expected to be available the longest according to an *oracle*, is selected to store the replica. This process

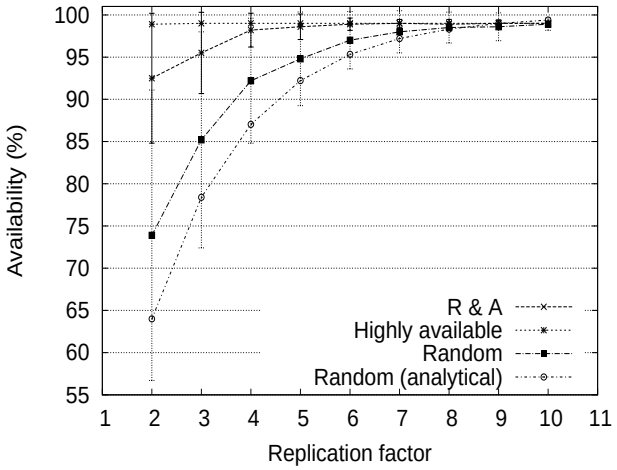

Figure 5: Availability using replication

is then iterated until all replicas have been stored.

In addition to compare our R&A strategy to practical random placement [4, 7, 27, 8] as well as the highly available strategy we also plot the availability provided by the analytical model of random placement (often used in replica maintenance papers). This availability is plotted for each k (denoted A_k) given by:

$$A_k = \sum_{i=1}^{k} C_k^i (\bar{p})^i (1-\bar{p})^{k-i} \quad (1)$$

where $\bar{p} = \frac{1}{N}\sum_{x=1}^{N} p_x$ (p_x is the mean availability of node x on the period and N is the number of nodes participating in the system). In the Skype trace we measured $\bar{p} = 0.4$.

Results. First of all, the highly available strategy unsurprisingly leads to a near-perfect availability even for very low replication factor. However, as explained in Section 2.3, load balancing is hurt. Turning to more realistic strategies we observe that whereas both R&A and random placement tend to achieve the same availability with a high replication factor, R&A placement leads to an increased availability mean compared to a random placement strategy up to $k = 9$. Replicating the whole dataset more than 10 times is highly unrealistic in any practical system, therefore our method represents a significant improvement over random strategy. For example, the same availability and standard deviation ($98.6\% \pm 3.4$ for random, $98.7\% \pm 2.4$ for R&A) are obtained with 8 replicas with a random strategy whereas only 5 replicas are required using our method.

Finally, we observe that analytical availability, despite showing the correct trend, is under-estimated. This illustrates that in a practical heterogeneous system, availability might not be reduced to a simple arithmetic mean. In fact equation (1) ignores all distributional information on the availability of the nodes, as it is reduced to the scalar value $\bar{p}$. The drawbacks of this definition of availability are explained in more detail in [11].

4.1.3 Load balancing

One of the main drawbacks of biasing replica placement (as opposed to a uniform random strategy) is the potential impact on the load balancing. In order to remain as scalable as possible the load balancing should not be too impacted by the placement strategy.

Note that even a random placement scheme is impacted by the effective availability of nodes in the network as this impacts the uniformity of the random choice. Since the algorithm performs the replication on online nodes, highly available ones tends to be naturally contacted more than their less available counterparts. This results in a slight skew in distribution (also observed in [31]). Random placement is an important standard to compare against for it is simple and efficient *wrt* load balancing.

Figure 6 plots the cumulative frequency of the number of replicas hosted by all nodes in the system, where random placement, highly available strategy and R&A are used. We note the critical skew when replicas are preferentially placed on highly available nodes. In fact less than 40% of nodes are burdened with 100% of the replicas. The load unbalance might eventually hurts the scalability of the system. resulting from creating clusters with R&A strategy is close to the random placement one. It could be explained by the fact that in addition to the random part of the heuristic, no behavior is given an advantage while computing anti-correlation, as opposed to a system where highly available nodes are always rewarded. Then if nodes availability patterns exhibit enough heterogeneity, R&A *naturally* balances the load, almost as well as the random placement strategy. Thus, R&A placement increases availability while balancing the load, providing a scalable alternative to existing solutions.

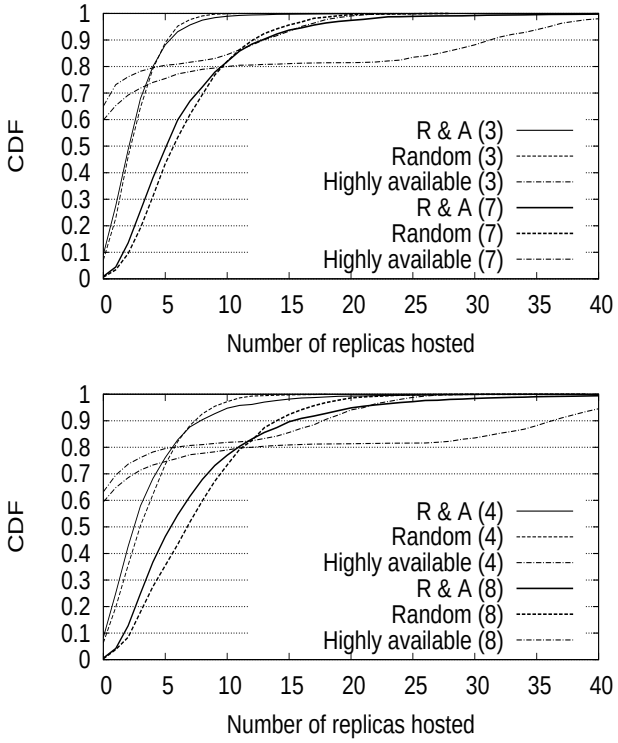

Figure 6: Load balancing for k=3;7 & k=4;8

4.1.4 R&A placement for erasure codes

Erasure codes have been proved to be another efficient way of obtaining data availability and durability in distributed storage systems [34]. We now assess the application of our R&A placement strategy when using such codes in place of replication.

So far, we have considered that the availability of a replicated data was achieved with at least one node holding a replica online at any time, among k, thus masking transient unavailability of other replicas. Erasure codes are able to reconstruct a data with j out of n encoded blocks. This can be considered as providing j available nodes out of a set of n. We thus seek to provide a subset j of nodes that maximize availability, so that the correct amount of encoded blocks can be found to reconstruct the data.

In practice, we model a typical j out of n redundancy scheme in which n code blocks are bunched in a cluster, and data is defined as available only if at least j code blocks are available within this cluster. The *code rate* is defined as j/n. The availability denoted $A_{(j,n)}$ is adapted from (1) and given by :

$$A_{(j,n)} = \sum_{i=j}^{n} C_n^i (\bar{p})^i (1-\bar{p})^{n-i} \quad (2)$$

We evaluate our R&A method using erasure codes on the same Skype trace, along with a random placement on n nodes. Figure 7 plots the availability obtained for three different *code rate* (1/2;1/3;1/4), with an increasing n. The top figure corresponds to data availability resulting from the real Skype trace while availability on the other one is computed analytically using equation (2) and $\bar{p} = 0.4$ as in part 4.1.2. We observe a clear improvement in data availability regardless of the *code rate* over the random placement scheme. For example, the same performance is obtained with a random selection and a code rate equal to 1/4 while using our method a *code rate* equal to 1/3 is sufficient. At the scale of the whole network, this 25% reduction saves an important amount of overhead. The intuition behind this result is that our placement strategy has a "built-in" notion of time span over a predefined period; each time the algorithm picks a random node and its anti-correlated counterpart, availability is increased on this period, while random placement relies on "luck" to pick nodes that will fill temporary unavailability holes. As we look for a larger set j of nodes online at the same time (whereas with basic replication, $k = 1$ replica is enough), the positive effect of a clever placement is increased. Moreover, this j among n availability target can also have other fields of application, such as for instance providing higher throughput when downloading a data from the storage system, as j replicas are available for parallel downloads (see *e.g.* [12]).

An interesting effect can be observed on both analytical and simulation figures. While for reasonable rates, curves show good availability, the rate 1/2 causes performance to degrade significantly as n increases, for both placement schemes (even if R&A still performs better). This is explained by

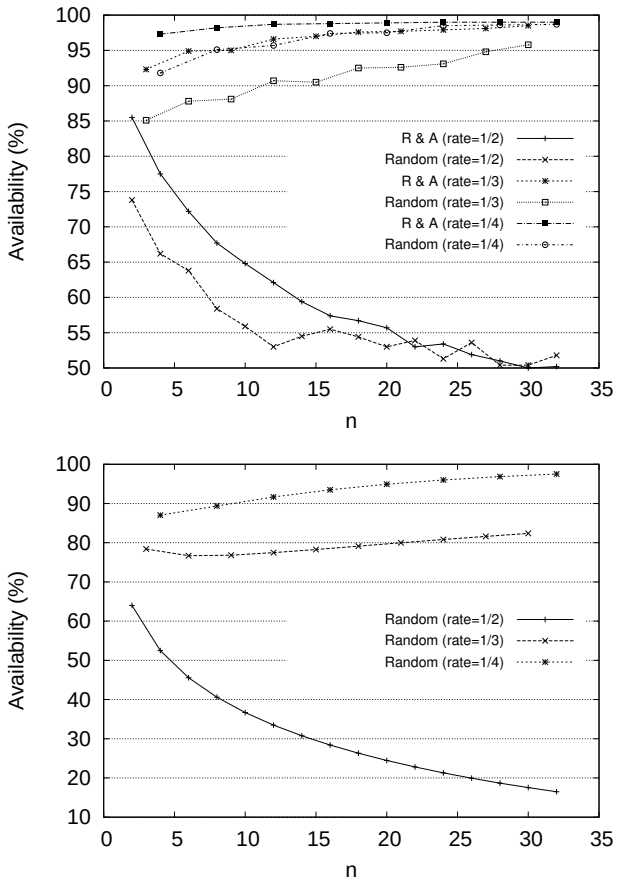

Figure 7: Availability using erasure codes in simulation (top) and analytically(bottom)

the fact that in this system (*i.e.* in this availability trace) the mean availability ($\bar{p} = 0.4$) is below the 1/2 code rate value. More details concerning this effect, and the relation between the mean availability and the code rate can be found in [20].

As in Section 4.1.2, we note that whereas analytical results provide tendencies about variations of availability with an increasing n, data availability values are under-estimated in all cases. This results from too restrictive a definition of the availability in Equation (2). More details can be found in [11].

5. TIMEOUT FOR REPAIRS

In addition to the initial replica placement strategy, a distributed storage system has to provide a reliable repair mechanism over time so as to ensure data durability. Bandwidth is a crucial element since data durability can only be ensured if there is sufficient bandwidth to repair lost replicas. There is a direct relation between a repair and the bandwidth consumed to perform it. In the following, we evaluate the bandwidth consumption as the number of repairs.

When designing a repair mechanism, the crucial challenge is to decide *when* to trigger a repair. In other words, how to distinguish permanent failures requiring a repair from temporary disconnections. In the latter case, a repair may turn out to be useless, consuming some bandwidth unnecessarily. A common way to decide on the nature of a failure is to use a *timeout*, a practical solution in asynchronous settings. Typically, once a node has not replied to a solicitation after a timeout has expired, it is considered as permanently failed. However deciding on a timeout value is difficult. More specifically, nodes may exhibit various availability patterns; therefore, defining a system-wide timeout value might not be optimal. Deciding on an optimal value for an administrator is also a tedious and difficult task.

We design an adaptive per-node timeout mechanism that solves both issues at once. This approach automatically sets a timeout at a node granularity, leveraging the availability history of each node in the system.

5.1 Per-node & adaptive timeout

This section describes how the timeout value is made both adaptive and per-node, (*i.e.* tailored to every single node based on its availability history). Remember that in a cluster, all nodes are monitored. Therefore for each node disconnection (*i.e.* the beginning of an unavailability session), a timer is started and runs until reconnection (either on a monitoring server, or by cluster nodes themselves, as explained in Section 3).

The cluster is then aware of the unavailability duration of each disconnected node. As in the classical timeout model, if the unavailability period of a given node exceeds its timeout value, the node is considered as permanently down and the system provides this cluster with a new node to be used to *repair* the lost replica. As opposed to most approaches that set a system-wide timeout, we aim at determining an accurate timeout value reflecting each node's behavior with respect to availability. This tends to suggest that one might analyze node failure detection in a standard model of decision making under uncertainty. In fact the *a priori* probability for a given system is too coarse to provide usable information for each given node.

Contrarily, adding extrinsic information, such as the availability patterns of a given node, might help when observing a given unavailability time to evaluate the probability that this node will return, therefore that the disconnection (the failure) is transient. To illustrate our purpose, let us consider the following example: if a node, usually always available , is detected unavailable for a few hours, the probability that it will reconnect is low, and lower as time passes. On the contrary, let us consider a node subject to a diurnal availability pattern. If such a node is detected unavailable for a few hours, the probability that it will reconnect is high. Hence, considering statistical knowledge at a node granularity can be useful to determine the probability that a node will reconnect (i.e. the failure is transient). This motivates the need for *per-node* timeouts in heterogeneous systems.

In systems implementing *reintegration*, it may be the case that a node, wrongfully declared as permanently failed, is reintegrated after the repair has taken place. This yields the presence in the cluster of more replica than the k required. In our approach we propose to *adapt* each node's timeout

dynamically to account for such situations. In case of an excess of replicas, this is translated in our adaptive method by setting a less aggressive timeout for each node in the cluster.

In the following we describe our per-node adaptive timeout mechanism.

5.2 Timeout model

The novelty of our timeout model is to be constructed at the node granularity. The model described below is then defined for each node, depending on its **own** attributes.

After the failure of a node, we call H_0, the event representing the fact that the node will return into the system and H_1, the event representing the fact the node will not return into the system. H_1 and H_0 are two disjoint events then $\Pr(H_1) = 1 - \Pr(H_0)$. $\Pr(H_0)$ is the *a priori* probability that the node will return into the system. $\Pr(H_0)$ is evaluated as the ratio between transient failures and all failures (either transient or permanent) in the system directly computed on a server. Note that this *a priori* probability could also, for example, be estimated from an aggregation protocol in a decentralized way. Let $t_{downtime}$ be the duration of the current downtime of the node and $t_{timeout}$ the timeout value associated to this node. Let its downtime distribution be f_d; this distribution verifies :

$$\int_{(t=0)}^{\infty} f_d dt = 1$$

then

$$\Pr(t_{downtime} > t_{timeout} \mid H_0) = \int_{(t=timeout)}^{\infty} f_d dt$$

By definition $\Pr(t_{downtime} > t_{timeout} \mid H_0) \in [0,1]$ and $\Pr(H_0) \in [0,1]$. $\Pr(t_{downtime} > t_{timeout} \mid H_0)$ then corresponds to the statistical knowledge on each node behavior. In fact as the availability history of each node is stored, its downtime probability distribution can be computed directly, and then also $\Pr(t_{downtime} > t_{timeout} \mid H_0)$ depending on the timeout value. By definition:

$$\Pr(t_{downtime} > t_{timeout} \mid H_1) = 1$$

$$\Pr(t_{downtime} < t_{timeout} \mid H_1) = 0$$

If the node does not come back into the system, on the one hand its downtime will be superior to any timeout value, on the other hand its downtime cannot be inferior to any timeout value. We also note:

$$P_{FA} = \Pr(H_0 \mid t_{downtime} > t_{timeout})$$

P_{FA} is called the False Alarm probability. P_{FA} represents the probability that a node comes back in the system while the system has decided that it would not (i.e. the node has been timed-out). The higher the P_{FA}, the more (probably) useless reparations are tolerated. According to Bayes' Theorem :

$$P_{FA} = \frac{\Pr(H_0 \cap (t_{downtime} > t_{timeout}))}{\Pr(t_{downtime} > t_{timeout})}$$

$$P_{FA} = \frac{\Pr(H_0) \times \Pr(t_{downtime} > t_{timeout} \mid H_0)}{\Pr(t_{downtime} > t_{timeout})}$$

$$P_{FA} = \frac{\Pr(H_0) \times \Pr(t_{downtime} > t_{timeout} \mid H_0)}{[\Pr(H_0) \times \Pr(t_{downtime} > t_{timeout} \mid H_0)] + 1 - \Pr(H_0)}$$

Finally :

$$P_{FA} = \frac{\Pr(H_0) \times \Pr(t_{downtime} > t_{timeout} \mid H_0)}{1 + \Pr(H_0) \times [\Pr(t_{downtime} > t_{timeout} \mid H_0) - 1]} \quad (3)$$

The definition interval of P_{FA} is then $[0, \Pr(H_0)]$.

To sum up, Equation (3) expresses the false alarm probability as a function of the *a priori* probability that a node will eventually come back into the system and of its downtime distribution. This expression is necessary to compute per-node timeouts, as explained in the next section.

5.3 Computing an adaptive per-node timeout

The scheme presented hereafter is performed in each cluster at each time unit (order of hours or days, depending on the stability of the system).

Each cluster has to deal with its departed and returning nodes. Returning nodes reintegrate their cluster. On the contrary, if a node is timed-out and if the cluster size falls under the replication factor, the system provides this cluster with a new node in order to create a new replica. This new node can be chosen randomly or following a specific placement strategy as proposed in the first part of this paper. Periodically, the following steps are performed within each cluster:

1. The cluster is updated so as to reintegrate wrongfully timed-out nodes and insert new nodes to compensate the removal of timed-out nodes;
2. The false alarm probability is computed (at the cluster level);
3. The timeout of each node in the cluster is computed.

The false alarm probability varies depending on how critical the situation is *wrt* replicas. We propose hereafter a method used to evaluate this situation (note that this method is not proved as being an optimal one, but just a practical example that we use for evaluation). At each time unit, we compare the current number of available replicas against the one on the previous week. This difference, noted Δ, is positive if there are more replicas currently than in the past, negative otherwise. Note that the only parameter measured is the number of available replicas, regardless of which node stores them. If there is no difference, the false alarm probability is defined as the half of its interval so $P_{FA} = \Pr(H_0)/2$. We define the step of variation as $\Pr(H_0)/k$, with k the number of desired replicas of data. P_{FA} is computed as: $P_{FA} = \Pr(H_0)/2 - (\Delta \cdot \Pr(H_0)/k)$. At the end of this step, each cluster has a false alarm probability that is a function of the measured criticality of the situation.

Once the false alarm probability has been determined in each cluster, each node is able to specify its own timeout value. According to Equation (3) we have :

$$\Pr(t_{downtime} > t_{timeout} \mid H_0) = \frac{P_{FA} \cdot (1 - \Pr(H_0))}{\Pr(H_0) \cdot (1 - P_{FA})}$$

$$\int_{(t=timeout)}^{\infty} f_d dt = \frac{P_{FA} \cdot (1 - \Pr(H_0))}{\Pr(H_0) \cdot (1 - P_{FA})} \quad (4)$$

Therefore the timeout value is set so as to solve Equation (4).

5.4 Evaluation

We evaluate our per-node timeout on public traces. We compare the cost/availability trade-off of our approach against systems using global timeout from aggressive (low) to relaxed (high) values. In order to evaluate the impact of the timeout in isolation from the replica placement strategy, we rely on a random placement scheme. Replicas of false positive nodes are also reintegrated in the global timeout simulation. We evaluate the trade-off on three different system traces, namely PlanetLab, Microsoft and Skype. As in Section 4.1.2 nodes having an uptime below 1% have been removed from the trace, resulting in a number of alive nodes respectively equal to 308, 51313 and 1901. The simulation is performed on a three-week period. The *learning period* is the first week, after which the history window simply slides. In fact an initial availability history must be available so that each node is able to compute its initial downtime distribution. Data availability mean and number of repairs are then evaluated during the next two weeks. During these two weeks downtime distributions of each node are updated following their behavior, after each new end of a downtime session.

Each of the *alive* nodes stores one data item, replication factor ranges from 3 to 6 (5 to 8 for Skype). We evaluate our approach and the global timeout one along the following metrics: mean data availability and number of repairs generated by timed-out nodes. Results for Microsoft and PlanetLab traces are provided in Figure 8 and Table 1. The Skype results (Figure 9 and Table 2) are given in the next section, as we will finally add results from the combination of the R&A placement strategy and the use of per-node timeout.

In Figures 8 and 9, the X-axis represents the mean of data unavailability in percent (*i.e.* $1 - availability$). The Y-axis is the number of repairs per data item and per day, it is equivalent to the number of timed-out nodes, as each of them triggers a repair. Then unavailability versus the number of repairs is plotted for various values of global timeout (10 to 80 hours for Microsoft and 20 to 280 hours for PlanetLab) and for the per-node timeout, in the classic way of representing timeouts versus repairs, as done in recent papers [30, 35].

Table 1 and 2 summarize the savings on the number of repairs when using our adaptive timeout, compared to global ones. These savings are given in percent and evaluated for equivalent availability which is linearly interpolated for global timeouts and for each replication factor. For instance on the Microsoft trace and for $k = 3$ (Figure 8 (bottom)) the interpolated point has coordinates (0.4 , 0.075). For an equivalent unavailability of 0.4% the global timeout would lead to 0.075 repairs by object and by day whereas our per-node timeout only need 0.063 , thus resulting in a 16% saving. Note that values have been rounded in the example and exact computations are given in Tables 1 & 2.

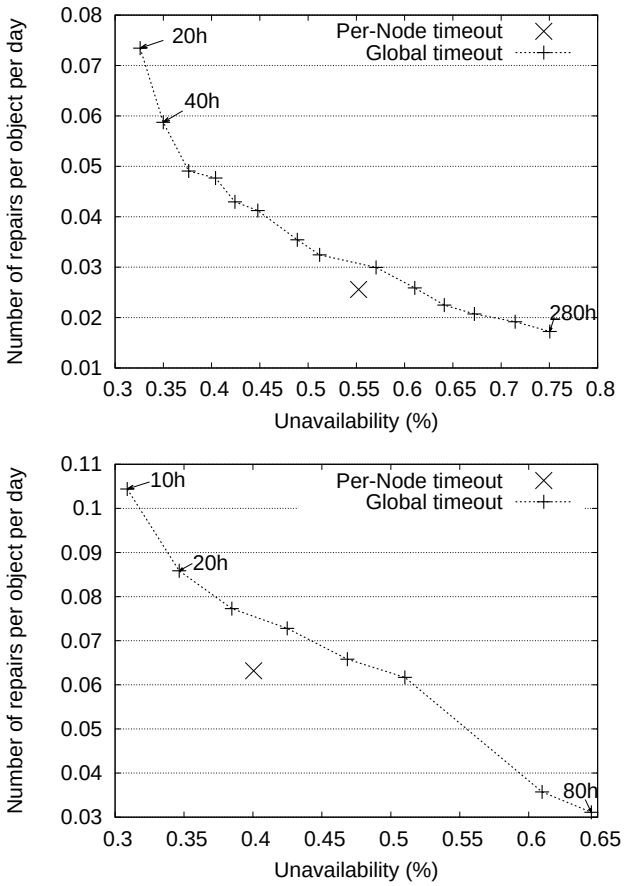

Figure 8: PlanetLab (top) & Microsoft (bottom) results for $k = 3$.

	3	4	5	6	mean
Microsoft	16.3%	17.9%	19.1%	20.7%	18.5%
Planet Lab	16.8%	23%	62.8%	48.4%	37.7%

Table 1: Reduction of the number of repairs

Results. Unsurprisingly a first observation applicable to all plots is that an aggressive global timeout value (10H) leads to a low unavailability but produces a high repair rate, triggering an important bandwidth consumption in practice. On the other hand, a relaxed timeout value (80H) enables to reduce the number of repairs at the expense of decreased data availability.

A second observation is that regardless of the global timeout value, our per-node timeout always provides a better trade-off between availability and the number of repairs. In other words, in all cases for an equivalent availability (and obviously for the same replication factor), the number of repairs will always be higher using global timeout than our per-node timeout. Note that both traces show various levels of node predictability (especially when using a one-week prediction period) [22]. Our approach thus saves significant bandwidth, resulting from the decreased number of repairs.

In addition, we emphasize that not only raw performance on

those metrics is greatly improved, but another important benefit of the adaptive timeout method is that a system administrator does not have to arbitrarily set a static value for timeout. The system self-organizes to compute adapted values, thus suppressing the need for such a decision before the runtime of the storage application.

6. COMBINING R&A AND ADAPTIVE PER-NODE TIMEOUT

So far, we have proposed and evaluated two techniques, leveraging the availability history of nodes. In this section, we consider their combination, for possible improvements.

As their respective merits have been presented separately, we now use both R&A placement and adaptive per-node timeout in the same simulation setting. The experiments are conducted on the Skype trace with adaptive and per-node timeout, while adding the R&A placement strategy to constitute clusters. Note that results on Planet Lab and Microsoft traces are not presented here as the random placement strategy already provides high availability on these traces, due to their high availability means (Figure 1). Note that R&A achieves similar performance. Deploying another placement strategy is thus of no interest. As in 5.4 the simulation is performed on a three week period. The *learning period* lasts a week. In fact, an initial availability history must be accessible so that *(i)* the system is able to establish anti-correlation on behaviors, and *(ii)* each node is able to compute its initial downtime distribution. In practice, traditional random placement could be used during the first week, so the storage system can be operational while the leaning phase is processed. The mean data availability and the number of repairs are then evaluated during the next two weeks.

Results are plotted in Figure 9 for a replication factor $k = 5$; other results are then summarized in Table 2. The comparison is made between our per-node timeout and a global timeout varying from 10 to 80 hours with a random placement strategy. Again, to ensure a fair comparison, reintegration of replicas (then hosted by returning nodes) is also included in the global timeout scheme. Finally, we plot mean unavailability and the number of repairs resulting from the combination of the R&A placement strategy and the per-node timeout in order to measure the additional gain while applying both our availability-based methods.

Results show that the combination of our two methods clearly outperforms global timeout and random placement. In fact even when compared with the most aggressive global timeout value (10H), availability is slightly increased regardless of the replication factor while the repair rate is greatly reduced by 37% on average in comparison to this most aggressive timeout approach.

7. DISCUSSION

Before concluding, we briefly discuss two other aspects of our proposal, namely cost and applicability.

Cost of the methods. A practical question is whether or not such availability-based methods imply significant costs

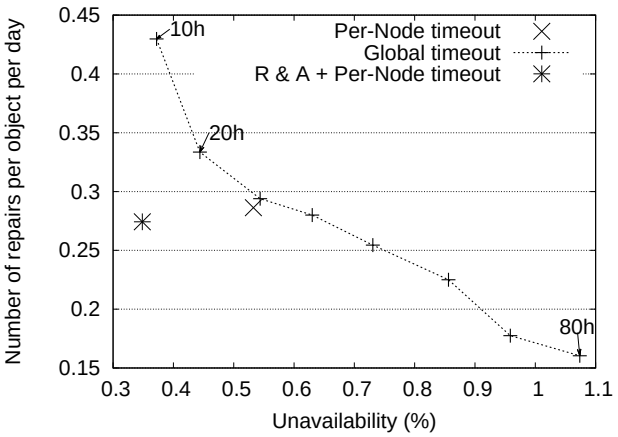

Figure 9: Results of R&A placement combined with adaptive timeout on Skype trace ($k = 5$).

	5	6	7	8	mean
Random	4%	9.5%	7.9%	10.7%	8%
R&A	36.2%	37.2%	37.5%	38%	37.2%

Table 2: Reduction of the number of repairs for different placement strategies, both using per-node timeout

for their implementation in a large-scale storage system. Such a framework requires operations that appear to be lightweight in terms of bandwidth and control message overhead: for monitoring, simple PINGs, in order to track other cluster nodes availability can be used. Finding anti-correlated nodes can be achieved in a scalable and inexpensive manner through a gossip mechanism (see T-Man for example) in a pure P2P system, while in a hybrid approach, the central server basically sorts its list of participating nodes. Finally, local operations on vectors are only CPU consuming. This is to compare with the replication and repair costs of distributed storage systems. The amount of data to store in modern systems, as well as the size of data files are constantly increasing. We sketch a basic example based on results from Table 2, where each of the 1901 nodes participating, backups $1GB$ of data; the system has to handle around $15TB$. A 38% mean saving of repairs per-node per-day represents around $4TB$ of bandwidth saved per-day at the network scale. In this light, network costs required to implement our techniques are marginal, especially when the network or the data stored grows large.

Applicability of our proposal. It is also important to note that the application of our mechanisms to availability issues is an illustration, and that their design goes way beyond. Other systems may leverage our approach, such as systems where activity patterns of some kind exist and can be predicted. It is for example known that workloads in enterprise data centers may exhibit patterns [33]; accurate workload placement may then save computation time if tasks are attributed to idle servers while known one are busy. Timeouts that are adapted to recurrent load of servers can help to distinguish between a crashed and a loaded server. This may

then constitute straightforward application of this work.

8. RELATED WORK

Studies on availability have motivated the idea of biasing the replica placement strategy, instead of using a random one. In [22], authors propose to store replicas towards highly available nodes. The work in [25] suggests offering a data availability proportional to the node stability. In paper [18], replica placement is biased towards nodes that have similar availability patterns, typically available at the same time. Very recently, paper [28] analyses replica placement strategies which optimize availability, "patching" the time by selecting highly available nodes when some time slots are not covered by at least one replica.

With regard to the distinction between transient and permanent failures some authors have very recently proposed advanced timeout design. The authors in [26] analyze the use of timeout in a stochastic model and compare solutions with reintegration against standard ones. The approach of [30] is the closest to ours. However their proposed algorithm mainly focuses on the accuracy of instantaneous detection but does not consider the impact of the false positive or false negative errors on average data availability and replication cost. The authors of [35] are the first to model node behavior with a continuous semi-Markov chain, which does not require the use of exponential distribution. However, they always assume a scalar value for availability, and a homogeneous behavior for all nodes in the system.

9. CONCLUSION

In this paper, we demonstrate the interest of leveraging the knowledge of node availability patterns, in particular when node exhibit heterogeneous behavior *wrt* uptime. While this idea has already been explored at a general level, we have advocated an implementation at a finer granularity, namely on a per node basis. We have addressed two important faces of distributed storage systems, namely replica placement and timeouts for repairs. Our evaluations conducted on real traces have shown substantial gain of this availability-aware framework. We expect those two practical contributions to be used as plug-ins in deployed systems.

10. REFERENCES

[1] Repository. http://www.cs.uiuc.edu/homes/pbg/availability/.

[2] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A five-year study of file-system metadata. In *ACM Transactions on Storage*, volume 3, New York, NY, USA, October 2007. ACM.

[3] Ranjita Bhagwan, Stefan Savage, and Geoffrey Voelker. Understanding availability. In *IPTPS, Int'l Work. on Peer-to-Peer Systems*, 2003.

[4] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: system support for automated availability management. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[5] Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: pick two. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[6] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *SIGMETRICS '00: Proceedings of the 2000 ACM international conference on Measurement and modeling of computer systems*, pages 34–43, New York, NY, USA, 2000. ACM.

[7] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, San Jose, CA, May 2006.

[8] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM.

[9] John R. Douceur. Is remote host availability governed by a universal law? *SIGMETRICS Perform. Eval. Rev.*, 31(3):25–29, 2003.

[10] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *SIGMETRICS '99: ACM international conference on Measurement and modeling of computer systems*, SIGMETRICS '99, pages 59–70, New York, NY, USA, 1999. ACM.

[11] Richard J. Dunn, John Zahorjan, Steven D. Gribble, and Henry M. Levy. Presence-based availability and p2p systems. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 209–216, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Abdullah Gharaibeh and Matei Ripeanu. Exploring data reliability tradeoffs in replicated storage systems. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 217–226, New York, NY, USA, 2009. ACM.

[13] Kevin M. Greenan, Parascale Inc, James S. Plank, and Jay J. Wylie. Mean time to meaningless: Mttdl, markov models, and storage system reliability. In *The 2nd Workshop on Hot Topics in Storage Systems (HotStorage2010)*, 2010.

[14] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the skype peer-to-peer voip system. In *Proceedings of the 5th international workshop on peer-to-peer systems (IPTPS '06)*, 2006.

[15] Andreas Haeberlen, Alan Mislove, and Peter Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 143–158, Berkeley, CA, USA, 2005. USENIX Association.

[16] Hai Huang, Wanda Hung, and Kang G. Shin. Fs2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, SOSP '05, pages 263–276, New York, NY, USA, 2005. ACM.

[17] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[18] Stevens Le Blond, Fabrice Le Fessant, and Erwan Le Merrer. Finding good partners in availability-aware p2p networks. In *Stabilization, Safety, and Security of Distributed Systems*, volume 5873 of *Lecture Notes in*

Computer Science, pages 472–484. Springer Berlin / Heidelberg, 2009.

[19] Sergey Legtchenko, SÃl'bastien Monnet, Pierre Sens, and Gilles Muller. Churn-resilient replication strategy for peer-to-peer distributed hash-tables. In *Stabilization, Safety, and Security of Distributed Systems*, volume 5873 of *Lecture Notes in Computer Science*, pages 485–499. Springer Berlin / Heidelberg, 2009.

[20] W. K. Lin, D. M. Chiu, and Y. B. Lee. Erasure code replication revisited. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, P2P '04, pages 90–97, Washington, DC, USA, 2004. IEEE Computer Society.

[21] Pietro Michiardi and Laszlo Toka. Selfish neighbor selection in peer-to-peer backup and storage applications. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 548–560, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] James W. Mickens and Brian D. Noble. Exploiting availability prediction in distributed systems. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 6–6, Berkeley, CA, USA, 2006. USENIX Association.

[23] Ramsés Morales and Indranil Gupta. Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 20(4):446 –459, apr. 2009.

[24] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In José C. Cunha and Pedro D. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 432–441. Springer Berlin / Heidelberg, 2005.

[25] Lluis Pamies-Juarez, Pedro García López, and Marc Sánchez-Artigas. Rewarding stability in peer-to-peer backup systems. In *ICON 2008. 16th IEEE International Conference on Networks*, pages 1 –6, dec. 2008.

[26] Sriram Ramabhadran and Joseph Pasquale. Durability of replicated distributed storage systems. In *SIGMETRICS '08: Proceedings of the 2008 ACM international conference on Measurement and modeling of computer systems*, pages 447–448, New York, NY, USA, 2008. ACM.

[27] Antony I. T. Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.

[28] Krzysztof Rzadca, Anwitaman Datta, and Sonja Buchegger. Replica placement in p2p storage: Complexity and game theoretic analyses. *International Conference on Distributed Computing Systems*, 0:599–609, 2010.

[29] Kiran Tati, , Kiran Tati, and Geoffrey M. Voelker. On object maintenance in peer-to-peer systems. In *In Proc. of the 5th International Workshop on Peer-to-Peer Systems*, 2006.

[30] Jing Tian, Zhi Yang, Wei Chen, Ben Y. Zhao, and Yafei Dai. Probabilistic failure detection for efficient distributed storage maintenance. In *SRDS '08: Proceedings of the 2008 Symposium on Reliable Distributed Systems*, pages 147–156, Washington, DC, USA, 2008. IEEE Computer Society.

[31] Laszlo Toka, Matteo Dell amico, and Pietro Michiardi. Online data backup : a peer-assisted approach. In *P2P'10, 10th IEEE International Conference on Peer-to-Peer Computing, August 25-27, 2010, Delft, The Netherlands*, 2010.

[32] Vytautas Valancius, Nikolaos Laoutaris, Laurent Massoulié, Christophe Diot, and Pablo Rodriguez. Greening the internet with nano data centers. In *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 37–48, New York, NY, USA, 2009. ACM.

[33] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 28–28, Berkeley, CA, USA, 2009. USENIX Association.

[34] Hakim Weatherspoon and John Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. In *IPTPS*, 2002.

[35] Zhi Yang, Yafei Dai, and Zhen Xiao. Exploring the cost-availability tradeoff in p2p storage systems. In *ICPP '09: Proceedings of the 2009 International Conference on Parallel Processing*, pages 429–436, Washington, DC, USA, 2009. IEEE Computer Society.