Exploring heterogeneity of unreliable machines for p2p backup

Piotr Skowron, Krzysztof Rzadca
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw
Warsaw, Poland
Email: p.skowron@mimuw.edu.pl

***Abstract*—P2P architecture is a viable option for enterprise backup. In contrast to dedicated backup servers, nowadays a standard solution, making backups directly on organization's workstations should be cheaper (as existing hardware is used), more efficient (as there is no single bottleneck server) and more reliable (as the machines are geographically dispersed).**

We present the architecture of a p2p backup system that uses pairwise replication contracts between a data owner and a replicator. In contrast to standard p2p storage systems using directly a DHT, the contracts allow our system to optimize replicas' placement depending on a specific optimization strategy, and so to take advantage of the heterogeneity of the machines and the network. Such optimization is particularly appealing in the context of backup: replicas can be geographically dispersed, the load sent over the network can be minimized, or the optimization goal can be to minimize the backup/restore time. However, managing the contracts, keeping them consistent and adjusting them in response to dynamically changing environment is challenging.

We built a scientific prototype and ran the experiments on 150 workstations in the university's computer laboratories and, separately, on 50 PlanetLab nodes. We found out that the main factor affecting the quality of the system is the availability of the machines. Yet, our main conclusion is that it is possible to build an efficient and reliable backup system on highly unreliable machines (our computers had just 13% average availability).

***Keywords*—distributed storage, enterprise backup, data replication, unstructured p2p networks, availability**

I. INTRODUCTION

Large corporations, medium and small enterprises, universities, research centers and common computer users are all interested in protecting their data against hardware failures. The most common approach is to keep backup copies on tape drives, specially designated storage systems, or to buy cloud storage space. All such solutions are highly reliable, but also expensive. In 2013 the costs of renting 1TB of cloud storage per year from Amazon, Google, Rackspace or Dropbox is approximately $1000. Additionally, for some organizations, internal data handling policies require that data cannot be stored externally. The price of a single backup server with raw capacity of 14TB often exceeds $12,000. A tape-based backup system for 14TB costs about $7,000. This figures do not include additional costs of service, maintenance and energy. With a large number of workstations that must be replicated at a single server, the server may become a bottleneck not offering satisfactory throughput; also, performance can be degraded by network congestion. More scalable solutions exist, but are even more expensive. Yet, the market for the backup solutions is vast. DataDomain, a company providing the modern backup systems, had in 2009 over 3.000 customers and over 8.000 systems deployed [30]. In the same year the company was bought by EMC for $2,4 billion.

There is still a need for cheaper alternatives for enterprise backup. On one hand, a significant research effort focuses on the optimization techniques for dedicated backup servers, such as deduplication techniques [13], [19]; or erasure codes [22], [25]. On the other hand, a p2p architecture can be explored in the context enterprise backup. Common PCs are cheaper than reliable servers. Also, in many cases, the unused disk space on the desktop workstations can be used without additional costs (Adya et al. [1] shows that the unused disk space on the desktop workstations is growing every year; the Moore's law for hard disks capacities, first formulated by Kryder [39] still holds). The bandwidth of the nodes connected in a distributed way scales better than of a single server; the load on the network is more evenly distributed causing less bottlenecks. The system can take advantage of the geographical dispersion of the resources, thus offering better protection in case of natural disasters (e.g., fire, flood) or theft. Finally, p2p solutions have already proved to work well in the enterprise environment (GFS [16], MapReduce [11], Astrolabe [27], DHT [10] used in HYDRAstor [15], etc.).

Indeed, many p2p storage systems have been already built [2], [5], [6], [8], [17], [18], [20], [21], [32], [38], [41]. The deduplication techniques get adapted for p2p storage systems [26], [40]. There are new erasure codes more suitable for p2p systems [22]. Finally, there are many theoretical models for data placement optimizing data availability [3], [4], [7], [14], [23], [28], [31] and backup/restore performance [24], [37]. However, real systems do not fully take advantage of the p2p architecture. There is a gap between theoretical models and real implementations. There are systems (e.g., OceanStore [17] and Cleversafe) that distribute data between geographically remote servers. These systems could be used for backup, but they both use dedicated servers, which stays in the contrast with our primary goal of creating cheap backup

system based on existing, unreliable machines.

There are p2p storage systems designed to work on unreliable machines; perhaps the most known such a system is Farsite [5] – a 6-years long Microsoft's project. However, Farsite offers much more than a simple backup. As a complete distributed file system, Farsite must deal with parallel accesses to data, must manage the file system namespace, and ensure that frequently accessed data is highly available. Such requirements force additional complexity and many architectural limitations that do not exist in case of backup system. On the other hand, since data backup is not a primary use-case, Farsite does not focus on implementing replica placement strategies (e.g. geographical dispersion of replicas, ensuring that data are backed up within a given time window etc.).

Bridging the gap between many theoretical models [3], [4], [7], [14], [23], [24], [28], [31], [37] and prototype implementations, we asked the following question: Is it possible to implement various data placement strategies especially when machines are unreliable? Certainly, there are more challenges than in the case of centralized or highly-available systems. The machines' unreliability, and perhaps low availability, requires data locations to change dynamically. Is it difficult to continuously optimize data placement with such assumptions? And, finally, is it difficult to take advantage of the machines and network heterogeneity?

Our contribution is the following: (i) We present an architecture of a prototype storage system that uses pairwise (bilateral) replication contracts for storing data. (ii) We show that we can efficiently manage the contracts and ensure efficient backup even under significant peers' unavailability. Our scientific prototype is evaluated in a real distributed environment.

We built a scientific prototype that replicates user data on different workstations of the organization. In our prototype, the machines that enter the system besides the standard activities also keep replicas of data of other machines. We assume that the workstations are heterogeneous and prone to failures, in particular: (i) hardware might be heterogeneous and inefficient; (ii) the workstations may have variable amount of unused disk space (the space that is available for keeping replicas); (iii) the workstations are not always available – computers may stay powered on, or be powered off when not used by anyone (transient failures) (iv) they may experience permanent failures after which it is not possible to recover data stored on a machine.

In contrast with fixed data placement policies (storing data in a DHT [2], [6], [20], [21], [41]), our replication is based on storage contracts between an owner of the data and its replicators. A contract for storing a data chunk of the owner i on the replicator j is a promise made by j to keep i's data chunk for a certain amount of time. Until the contract expires, it cannot be dropped by j; but it can be revoked by i. Since every data chunk is associated with a list of storage contract, each chunk can be placed at any location (the location depends on the placement strategy). This contract-based architecture can be exploited in two ways. First, the contracts form an unstructured, decentralized architecture that enables to optimize replica placement, making the system both more robust and able to take advantage of network and hardware configuration. Second, contracts also allow strategies for replica placement that are incentive-compatible, such as mutual storage contracts [9], [31]. To the best of our knowledge, all previous literature on mutual contracts focuses on theoretical analysis only. We complement these theoretical works, by presenting an architecture of a contract-based storage system. Yet, in this paper, for the sake of concreteness, we focus on optimization of replica placement for p2p backup in a single organization, where incentives are not needed.

Our prototype (with the source code) is available for download with an open-source lincense at http://www.mimuw.edu.pl/~krzadca/nebulostore. We tested our prototype on 150 computers in students' computer laboratories; and on 50 machines in Planet-Lab. The lab environment might be considered as a worst-case scenario for an enterprise network, as the computers have just 13% average availability and are frequently rebooted. Moreover, we assumed that all the local data is modified daily.

The results of our work show that: (i) in a p2p backup system we are able to efficiently transfer data chunks – the bandwidth of such a system scales linearly with the number of machines. (ii) Even on machines with very low availability we are able to efficiently optimize placement of the replicas. We verified two different placement strategies (where the optimization goal was either to finish the backup of each data chunk within a given time, or to enforce a certain geographical disperison of the replicas). This leads to our main conclusion: (iii) It is possible to create an efficient p2p backup system and to take advantage of resources' heterogeneity. (iv) Hardware unavailability has a significant impact on the performance of the backup; because of unavailability the time needed for direct communication of two peers can be long (on average 20h). We call this effect the *cost of unavailability*. Our measurements confirm the simulation results of Sharma et al. [33] and Tinedo et al. [35].

Since our results are supported not only by the simulations, but also by measurements of an implementation on a real system, we consider them as the proof of the concept that an efficient p2p backup systems can be created and that the heterogeneity of the machines in such a system can be explored.

II. Related Work

HYDRAstor [15] and Data Domain [42] are commercial distributed storage systems, which use data deduplication to increase the virtual disk space.

Many papers analyze various aspects of p2p storage by either simulation or mathematical modeling. Usually, the anal-

ysis focuses on probabilistic analysis of data availability in presence of peers' failures (e.g., [3]). Douceur et al. [14], similarly to our system, optimizes availability of a set of files over a pool of hosts with given availability: theoretical as well as simulation results are provided for file availability. Chun et al. [7] studies by simulation durability and availability in a large scale storage system. Bhagwan et al. [4] and Rodrigues and Liskov [28] show basic analytical models and simulation results for data availability under replication and erasure coding. Finding the schedule of the transfers which minimizes the restore time and analysis of the impact of the size of the set of the replicators on the restore time is described by Toka et al. [37]. Pamies-Juarez et al. [24] studies the impact of the redundancy on the data retrieval time. Our paper complements these works by, first, presenting the architecture that allows for implementing placement strategies; second, by considering other measures of efficiency; and, third, by proving that various optimization strategies can be accomplished in unreliable environment.

As the context of this work is data backup in a single organization, we do not analyze incentives to participate in the system. However, to store the data, our system relies on agreements (contracts) between peers. In contrast in DHT-based storage systems contracts are (implicitly) made between a peer and the system as a whole. Thus, our architecture naturally supports methods of organization that emphasize incentives for high availability, such as mutual storage contracts [9], [31] (also these using asymmetric contracts [23]). It is worth mentioning that some papers explore the social interconnections while choosing the replica locations [38]; the tradeoffs between redundancy, data availability and the ability to place data on the trusted nodes is analyzed by Sharma et al. [33] and Tinedo et al. [35]. These methods can be adopted in our system.

Many p2p file systems [2], [6], [20], [21], [41], use storage and routing based on a DHT [10], [29]. The address of the block, which is a hash of its content, fully determines the locations of its replicas. Thus, such architecture is less suitable for balancing the load on replicating workstations, or for optimizing the placement of replicas. While these solutions focus on consistency of the data being modified by multiple users, this paper focuses on the best replication of the data, which can be modified by its owner.

OceanStore [17] and Cleversafe propose to spread replicas to geographically remote locations. These systems combine software solutions with a specially designed infrastructure that consists of numerous, geographically-distributed servers. The main contribution of these systems, from our perspective, is the resignation from a common DHT and the introduction of a new assumption that any piece of data can be possibly located at any server. These systems, however, do not discuss the issue of replicating data on the ordinary workstations (which are, in contrast to the servers, frequently leaving and joining the network) and do not present any means allowing to handle such dynamism.

Wuala [18] provided distributed storage based based not only on a specially dedicated infrastructure, but also on a cloud of workstations of users who install Wuala application. However, since late 2011, Wuala no longer supports p2p storage. The idea of using a hybrid architecture of central servers and user machines, called in the context of backup as peer-assisted backup is also explored by Toka et al. [36]. Other p2p backup software include Backup P2P, Zoogmo, or ColonyFs.

FreeNet [8] is a p2p application that exposes the interface of a file system. Its main design requirement is to ensure anonymity of both authors and readers. The underlying protocol relies on proximity-based caching. When a data item is no longer used, it can be removed from a caching location. Similarly, in Pangea [32] a replica is created whenever and wherever a data is accessed.

Farsite [5] was a Microsoft's 6-years long project aimed at creating distributed file system for sharing data between thousands of users. The retrospective from the project [5] gave us the feeling of following a good direction. Firstly, the authors emphasize that real scalability must face the problem of constant failures in the network. Secondly, they claim that in a scalable system, manual administration must not increase with the size of the network; we followed the both requirements when formulating our hypothesis.

There are a few substantial differences between Farsite and our prototype implementation. Most importantly, Farsite's architecture does not rely on mutual contracts, which allow us to implement both incentives and mechanisms ignoring the black listed peers. Farsite is a distributed file system and many of its use cases cause the greater complexity of the system. On the other hand, as Farsite is not a backup system, it does not support backup-specific requirements like placing replicas in geographically distributed locations, optimization of the backup/restore time, etc.

III. System Architecture

Our system uses a mixed architecture that stores control information, meta-data and data in three different ways. The control information that allows peers to locate and to connect to each other must be located efficiently — hence we use a DHT as a storage mechanism. In contrast, each peer is responsible for finding and managing peers who replicate its data (its *replicators*). Such replication contracts enable us to optimize replica placement and thus to tune replication to a specific network configuration. The meta-data describing replication contracts are kept by both the data owner and the replicator. Chunks of data are kept in an unstructured overlay; concrete locations are described by the meta-data.

A. Control information

The basic attributes of a peer are kept in a structure called *PeerDescriptor*. For each peer, its PeerDescriptor contains:

- identification information (public key);
- information needed to connect to this peer (its current IP address and a port of an instance of our software running on a workstation) and user account name in the operating system (account name is required by the current implementation of data transmission layer — see Section III-C);
- identifiers of *synchro-peers* (see Section III-D).

PeerDescriptors of all peers are kept in a highly replicated DHT: peer's ID (a hash of its public key) is hashed to its PeerDescriptor. As the size of the control information is small, we are able to afford strong replication (compared to a generic DHT). Thus, instead of a single peer, many peers keep the data hashed to a part of the key-space.

B. Replication contracts

The main goal of our prototype is to support nontrivial replica placement strategies; we need to be able to store any replica at any peer. This architecture contrasts with content addressable storage systems that put each chunk of data under an address that is fully determined by the chunk's unique identifier (e.g. hash of its content). As a trade-off for flexibility of placing replicas at any location, we need a mechanism to locate data.

In our system each peer keeps information about replica placement of its data chunks in an index structure called *DataCatalog*. For each data chunk, the catalog stores information about: (i) identifiers of the peers that keep replicas of the data chunk (hereinafter *chunk replicators*); (ii) size of the chunk; (iii) version number of the chunk.

Additionally, each peer keeps information about data chunks it replicates. As each storage contract is kept in exactly two places (the owner and the replicator), contracts are consistent and it is easy to retrieve lost metadata (the DataCatalog). Because peers are unreliable, the process of contracts negotiation can break at any point, possibly leading to two types of inconsistency: an owner o believes j is its replicator, while j is not aware of such a contract; or a peer j believes to be o's replicator, while o is not aware of such a contract. Contracts negotiation is however idempotent and because the contracts are kept both by owner and replicator, such inconsistencies can be easily fixed. Each peer periodically sends messages to its replicators with the claimed contracts (and versions of data chunks, which allows the replicators to update out-of-date chunks). Each replicator periodically sends similar information to the appropriate owners. Detected inconsistencies can be resolved either by adopting the owner's state; or by always accepting a replication agreement.

The DataCatalog is persisted in a file but it is not replicated between peers. In this way we avoid an additional overhead of updating the catalog at remote locations, when the contract for any chunk is changed; because machines are unreliable, in many cases we even would not be able to update the DataCatalog at the remote peers as they can be simply unavailable. On the other hand, both owners and replicators are aware of all their storage contracts. When the local data of any peer is lost, the peer (the owner) gossips the information about the failure. The replicators answer the gossip message with the information about the contracts; the owner uses this information to rebuild the DataCatalog. Once the DataCatalog is reconstructed, the owner locates and rebuilds all missing data. Since the DataCatalog is persistent, its reconstruction is required only in case of non-transient failure; thus it does not cause much overhead.

Alternatively, in an enterprise environment where a (replicated) server is an affordable option, the meta-data can be kept centrally (in primary memory for faster access). This solution is, however, less scalable.

We designed the mechanism that is responsible for relocating or additionally replicating data chunks that are weakly replicated according to the given abstract metric. The specific metric used in our evaluation takes into account peers' availability, bandwidth and geographic distribution; it tries to keep all but one replicas as close as possible not to overload the network and to keep one replica in remote location for additional safety (for location-dependent failures, such as fire, flood, etc.) . The metric also balances the load on the machines so that each data chunk can be replicated within the required *backup window* (the time requirement for each chunk to be backed up). The precise metric is described in the full version of this paper [34]). The optimization mechanism is based on hill-climbing — in consecutive steps, each peer performs locally optimal changes of the contracts. The optimization can proceed even if large fraction of peers is unavailable; to perform a single step we require only 3 peers to be available. Thus the mechanism is suitable for unreliable environments.

When nodes parameters (e.g. availability) change, or when a large number of nodes is added to the system, the contracts are renegotiated. If each such change resulted in data migration, the network and the hosts could easily become overloaded. Therefore the process of changing a contract is more elaborate. The contracts are allowed to change frequently but such changes do not require data migration. Such temporary contracts are periodically (e.g., daily) committed; after a contract is committed, the data is migrated. The complete mechanism involves some additional details as it must take into account also possible communication failures (see [34]).

C. Data Transmission and Updates

Every member of the network, before placing its replicas at a remote peer, must obtain this peer's permission. Once the peers reach an agreement, they mutually authorize each other using identities (public keys) available in peers' *PeerDescriptor*s (stored in the DHT).

The data is transmitted in an encrypted connection. In the current implementation, we use standard Linux tools for data transfers. Each peer runs a ssh daemon that acts as a server

that accepts connections of data owners. A peer uses scp to transfer its data.

When an owner modifies its local copy, the updated chunk must be propagated to the network. The replicators are informed of the changed versions of the data chunks through periodic control messages (the versions numbers are attached to the messages containing contracts sent between the owner and the replicator, described in the previous subsection). The unavailable peers are informed about the changes of data chunks through asynchronous messages, described below. Once the replicator finds there is a new version of the data chunk it replicates, it downloads the new version either from the owner or from the other replicators. Note that if the data owner were responsible for uploading the new version to the replicators, a successful transfer would require both the owner and the replicator to be available. In our solution, the replicator is responsible for keeping replicas up to date so we only require that the replicator and any other replicator or the owner is available.

Unlike common backup systems, our system stores only the last version of each data chunk. A system storing many previous versions may be built in the same way as e.g. version control software (svn, git) uses a standard filesystem; more specifically, the previous versions (or the deltas) can be kept in the same data chunk; or the deltas can be kept in separate data chunks.

D. Asynchronous/Off-line Messaging

We assume that the workstations may be unavailable for some time just because they are temporarily powered off. In contrast to many distributed storage systems (e.g., GFS [16]), in such a case our system does not rebuild the missing replica immediately, in order not to generate unnecessarily load on other machines nor on the network. Instead, when the unavailable peer eventually joins back the network, it efficiently updates its replicas. To inform the unavailable peers about the new version numbers of their replicas and about the contracts, we use asynchronous messaging. The control messages sent to the peer that is currently unavailable are cached at, so called, *synchro-peers*. We use group communication to synchronize the messages within each (small) group of synchro-peers. As opposed to Defrance et al. [12], who present the mechanism of caching the messages on routers, we chose to design the concept of synchro-peers to limit the costs of the additional hardware.

An asynchronous message from i to j is sent to the *synchro-peers* of j. Synchro-peers is a set of peers, defined for every peer j (j is called in this context a *target peer*) that keep asynchronous messages for j. Synchro-peers of j include j, so every message will be delivered to the target peer by the same means as it is delivered to the other synchro-peers. Each synchro-peer periodically tries to send the asynchronous message to the synchro-peers that have not yet received the message; the IDs of synchro-peers that have not yet received the message are attached to the message (see [34] for details).

Using asynchronous messages has two advantages. First, the asynchronous message is delivered with high probability even when the sender is unavailable. Second, the data may be downloaded concurrently from multiple replicators.

IV. Experimental Evaluation of the Prototype

A. Experimental environment

We performed the experiments in two environments: (i) computers in the faculty's student computer labs; and (ii) the PlanetLab. We ran our prototype software for over 4 weeks in the labs and over 3 weeks in PlanetLab. Each computer acted as a full peer: it owned some data and also acted as a replicator. The data was considered as modified at the beginning of each day; thus each day we expected the system to perform a complete backup. If the transfer of a particular data chunk did not succeed within a day, the following day we transferred a newer version of the chunk. We used chunks of equal size – 50MB.

1) Students computer lab: We run our prototype software on all 150 machines of the students' computer lab. The availability pattern might be considered as a worst case scenario for an enterprise (or a pessimistic one for an academic system). The lab is open from Mondays to Fridays between 8:30am and 8pm and on Saturdays between 9am and 2:30pm (however we did not use this information in our placement strategies). The students frequently (i) switch off or (ii) reboot machines to start Windows; each day at 8pm the computers are (iii) switched off by the administrators (the machines are not automatically powered on the next day).

The amount of local data was sampled from the distribution of storage space used by the students on their home directories (scaled so that the average value was 3GB); the resulting distribution is similar to uniform between 0 and 8 GB.

The local storage space depended on machines' local hard disks; and varied between 10GB (50% machines), 20GB (10% machines), and 40GB (40% machines).

The computers in students lab have very low average availability (the median is equal to 13%). Figure 1 presents the distribution of the availabilities of the computers in lab. Figure 2 presents the distribution of the up time of the computers and the time between their consecutive availability periods within a single day (the nights are filtered out). Low availability coupled with long session times constitute a worst-case scenario for a backup application: in contrast to short, frequent sessions, here machines are rather switched on for a day, then switched off when the lab closes and then remain off during the next week.

2) PlanetLab: The experiments on PlanetLab were using 50 machines scattered around Europe. Each machine was provided 10GB of storage space and had 1GB of local data intended to be backed up. The machines were almost contin-

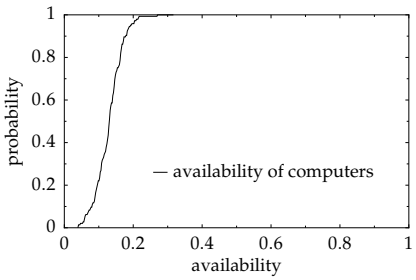

Figure 1. The cumulative distribution function of the availability of the computers in students lab.

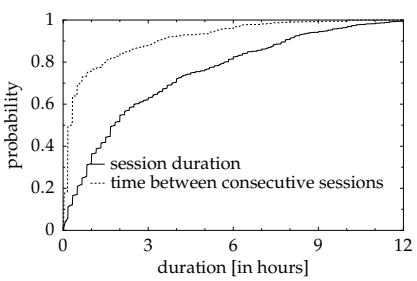

Figure 2. The cumulative distribution function of the session durations and the times between consecutive sessions for the computers in students lab (the nights are filtered out).

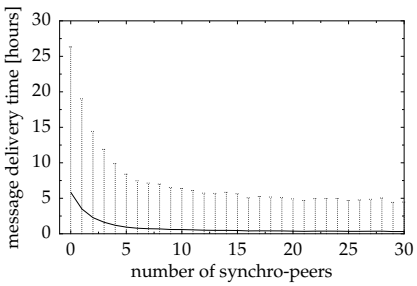

Figure 3. The dependency between the number of synchro-peers and the delivery time of the asynchronous message. Delivery time measured from the first time of availability of the receiver after the message was sent.

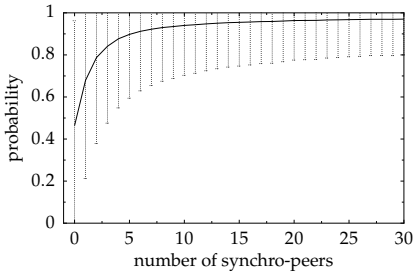

Figure 4. The dependency between the number of synchro-peers and the probability of delivery a message to any synchro-peer.

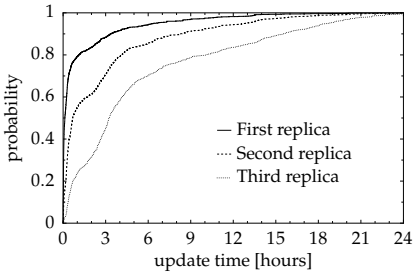

Figure 5. The cumulative distribution function for the time of creating a replica of data chunk. The time is relative to the data owner. Lab; data collected over 4 weeks of running time.

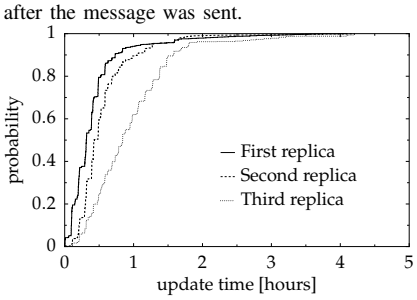

Figure 6. The cumulative distribution function for the time of creating a replica. The time is relative to the data owner. Planet-Lab; data collected over 3 days.

uously available (availability equal to 0.91: it is not 100% as some machines were overloaded for some time).

B. Asynchronous messages

In this subsection we present how the asynchronous messaging influence message delivery time and the probability that the message is delivered. For our analyzes we used the availability traces from the students' computer lab. We varied the number of synchro-peers per peer between 0 and 30. For each number of synchro-peers, we generated 100,000 messages with random source, destination and sent time.

Figure 3 presents the dependency between the number of synchro-peers and the delivery time of a message. Because the message delivery can be accomplished only when the receiver is active, we present the delivery time measured from the first availability of the receiver after the message was sent. Ideally, the message should be delivered just after the receiver goes online. The results show that synchro-peers significatly reduce delivery time measured from the perspective of the receiver. For the number of synchro-peers higher than 5, the advantage becomes less significant – 5 synchro-peers ensure low delivery time and induce low overhead to the system.

Figure 4 presents the dependency between the number of synchro-peers and the probability of a successful delivery a message to any synchro-peer. We are interested in calculating such probability because a message delivered to a synchro-peer is, in fact, a replica of the original message. Thus, synchro-peers should enable message delivery even in case of long term absence of the sender. The results show that synchro-peers significantly increase this probability – with 5 synchro-peers the system delivers 90% of the messages, while without synchro-peers, more than half of the messages are lost.

C. Placement strategies

1) Students' computer lab: The goal of tests on the labs was to verify how the system copes with low availability of the machines. We set the placement strategy to optimize the backup/restore time of data and we set the available bandwidth of the all machines to be equal. This reduced to the strategy that places on each peer the amount of data proportional to its availability (see full version [34] for more details).

During the first 3 days of experiments we measured the ratio: the total size of data replicated by a peer (in MB) to the peer availability. For each day, we considered only the peers that were switched on at least once. We also restricted the measurements only to peers with at least 9GB storage space (that could accommodate, on the average, 3 replicas), to separate the effect of insufficient storage space. The average values and the standard deviations of the ratios for the 3 days are presented in Table I. The standard deviation is low in comparison to the average (the deviations are 18%, 13% and 8% of the corresponding average) which shows that the replicas were distributed according to our expectations (the load was balanced correctly).

2) PlanetLab: The goal of PlanetLab tests was to verify how our system handles other placement strategies (including

TABLE I
THE RATIO: TOTAL SIZE OF REPLICATED DATA (IN MB) TO THE AVAILABILITY FOR THE FIRST 3 DAYS OF EXPERIMENTS IN THE LAB.

	Utility (weighted replicated data) [MB/avail.]	
day	**average**	**standard deviation**
1	34487	6086
2	60489	8141
3	69658	5496

specific geographic distribution requirements). We used the strategy that requires one replica to have TTL distance from the owner in range $\langle 3, 8\rangle$ and other replicas to be as close to the owner as possible. Additionally, to simulate the heterogeneity of the bandwidth of the machines, we set the limit on the bandwidth for half of the machines to 500 KB/s, and for the other half to 1000 KB/s. We set the backup window to 4500s. Each machine had the same amount of local data (1GB); the disk space limit was 4GB. We expected that the low-bandwidth machines will be less loaded than those from the high-bandwidth group. Assuming that machines are continuously available, a low-bandwidth machine should replicate at most 2.25GB; and a high-bandwidth machines at most 4.5GB.

There is a tradeoff between the backup time and the geographic distribution of data. We tested two parameter settings that assigned different weights to geographical distribution of replicas (see [34] for details). We used the parameter M that means that increasing the backup duration of a single chunk by M seconds is equally unwanted as increasing the TTL distance of this chunk by 1 outside of required range $\langle 3, 8\rangle$. As PlanetLab is highly geographically distributed, it is difficult to find machines with low TTL distances. For $M = 1$ the average TTL distance between the replica and the owner was equal to 11.6 (standard deviation equal 3.7). In this case only two machines exceeded their backup window (by at most 108 s). For $M = 100$, the replicas had better geographic distribution: mean TTL equals 8.1 (standard deviation equal 3.8). However, the backup duration was increased – 13 machines exceeded their backup window. The average excess of the backup window was equal to 222s (5% of the backup window) and the maximal 415s (9% of the backup window).

D. Duration of backup of a data chunk

We measured the time needed to achieve the consecutive redundancy levels (the required number of replicas) for each data chunk. The time is measured relative to the data chunk owner online time: we multiplied the absolute time by the owner's availability. We consider the relative time as a more fair measure because: (i) the transfer to at least the first replica requires the owner to be available; (ii) data can be modified (and thus, the amount of data for backup grows) only when the owner is available; (iii) we are able to directly compare results from machines having different availabilities. The distribution of time needed to achieve the consecutive redundancy levels is presented in Figure 5 (lab) and Figure 6 (PlanetLab).

1) Lab: The average time of creating the first, the second and the third replica of a chunk are equal to, respectively, 1.1h, 2.7h and 5.5h (the average time needed to create any replica is equal to 3.1h). We consider these values to be satisfactory as the average relative time for transferring a single asynchronous message holding no data (a message with no synchro-peers), calculated based on availability traces, is equal to 2.6h.

The maximal values, though, are higher: 24h, 29h and 32h. These high durations of replication are almost entirely the consequence of peers' unavailability. The maximal time needed to deliver an asynchronous message with 3 synchro-peers is of the same order (21.5h, measured relatively to source online time, see Section IV-B). Moreover, if we measure only the nodes with more than 20% average availability, the times needed to create the replicas are equal to 1h, 1.6h and 3h and maximal values are equal to 12h, 18h and 20h.

2) PlanetLab: The average times needed for creating the first, the second and the third replica are equal to, respectively, 0.5h, 0.7h and 1.1h. The maximal values are equal to 4.0h, 4.2h, and 4.2h. These values are significantly lower than in the case of the students' lab even though the distance between the machines is higher and the computers in students lab are connected with a fast LAN. This result once again proves that the unavailability of the machines is the dominating factor influencing the backup duration.

The average time needed for transferring a data chunk is equal to 0.76h. This corresponds to the throughput of 4.49Mb/s (Planet-Lab uses standard Internet connections).

V. CONCLUSIONS

We present an architecture of a p2p backup system based on pair-wise replication contracts. In contrast to storing the data in a DHT, in our approach the placement can be optimized to a specific network topology, which allows to take into account e.g. the geographical dispersion of the nodes. We implemented a prototype and tested it on 150 computers in our faculty and 50 computers in PlanetLab.

During implementation and initial tests we encountered numerous issues we did not expect: e.g., updating data catalog remotely whenever any contract is changed is highly inefficient; revoking the contracts cannot be done asynchronously; changing contracts too often is inefficient; each contract must be kept by both the data owner and the replicator and the two versions have to be kept consistent. We think that these problems should motivate others researchers to verify their ideas, in addition to simulations, by constructing prototype implementations.

Our most important result is that the backup time increases significantly if machines are weakly-available: from 0.76h for nearly always-available Planet-Lab nodes to 3.1h for our lab with just 13% average availability. This *cost of unavailability* makes some environments less suitable for p2p backup. The irregular environments negatively influence the maximal durations of data transfer. Choosing machines with better availability strongly reduces this effect (for instance, by restricting our lab environment to machines with more than

20% availability, the average backup time decreases from 3.1h to 1.9h). Moreover, in enterprise environments such irregular availabilities should not be the case. There, however, the machines may have their specific, regular availability patterns. In such case it may be valuable to use more sophisticated availability models.

Yet we must stress that it is possible to build an efficient and reliable backup system, even using weakly-available machines with irregular session times. We managed to run our prototype on 150 machines, showing that it is possible to take advantage of the heterogeneity of the p2p environment, in particular: the geographic dispersion of the machines, the network connections between the machines, different bandwidths, disk spaces and availabilities.

ACKNOWLEDGEMENTS

The research is funded by the Fundation for Polish Science "Homing Plus" Programme co-financed by the European Regional Development Fund (Innovative Economy Operational Programme 2007-2013. The authors thank Sonja Buchegger and Gunnar Kreitz for their helpful remarks.

REFERENCES

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[2] B. Amann, B. Elser, Y. Houri, and T. Fuhrmann. Igorfs: A distributed p2p file system. In *P2P, Proc.*, pages 77–78, 2008.

[3] S. Bernard and F. Le Fessant. Optimizing peer-to-peer backup using lifetime estimations. In *Damap Proc.*, 2009.

[4] R. Bhagwan, S. Savage, and G. Voelker. Replication strategies for highly available peer-to-peer storage systems. In *Fu-DiCo, Proc.*, LNCS. Springer, 2002.

[5] W. J. Bolosky, J. R. Douceur, and J. Howell. The Farsite project: a retrospective. *SIGOPS Oper. Syst. Rev.*, 41:17–26, April 2007.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[7] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *NSDI, Proc.*, volume 6, 2006.

[8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, pages 46–66, 2001.

[9] L. Cox and B. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *ACM SOSP, Proc.*, pages 120–132, 2003.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP, Proc.*, 2001.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[12] S. Defrance, A.-M. Kermarrec, E. L. Merrer, N. L. Scouarnec, G. Straub, and A. van Kempen. Efficient peer-to-peer backup services through buffering at the edge. In *P2P*, pages 142–151, 2011.

[13] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *FAST*, 2011.

[14] J. Douceur and R. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *DISC, Proc.*, volume 2180 of *LNCS*, pages 48–62. Springer, 2001.

[15] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. Hydrastor: a scalable secondary storage. In *FAST, Proccedings*, pages 197–210, 2009.

[16] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. In *ACM SIGOPS, Proc.*, volume 37, pages 29–43. ACM, 2003.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35:190–201, November 2000.

[18] T. Mager, E. Biersack, and P. Michiardi. A measurement study of the Wuala on-line storage service. In *P2P*, Sept. 2012.

[19] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *FAST*, 2011.

[20] J. michel Busca, F. Picconi, and P. Sens. Pastis: A highly-scalable multi-user peer-to-peer file system. In *Euro-Par, Proc.*, 2005.

[21] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31–44, 2002.

[22] F. E. Oggier and A. Datta. Self-repairing homomorphic codes for distributed storage systems. In *INFOCOM*, pages 1215–1223, 2011.

[23] L. Pàmies-Juárez, P. García-López, and M. Sánchez-Artigas. Enforcing fairness in P2P storage systems using asymmetric reciprocal exchanges. In *P2P*, pages 122–131, 2011.

[24] L. Pamies-Juarez, P. G. Lopez, and M. S. Artigas. Availability and redundancy in harmony: Measuring retrieval times in p2p storage systems. In *P2P*, pages 1–10, 2010.

[25] L. Pamies-Juarez, F. E. Oggier, and A. Datta. Decentralized erasure coding for efficient data archival in distributed storage systems. In *ICDCN*, pages 42–56, 2013.

[26] O. Papapetrou, S. Ramesh, S. Siersdorfer, and W. Nejdl. Optimizing near duplicate detection for p2p networks. In *P2P*, pages 1–10, 2010.

[27] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21:2003, 2001.

[28] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *IPTPS, Proc.*, volume 3640 of *LNCS*. Springer, 2005.

[29] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, 2001.

[30] D. Russell. Emc acquires data domain, becomes deduplication leader, signals deduplication as a "must have" capability. Gartner Research, ID Number: G00170233, 2009.

[31] K. Rzadca, A. Datta, and S. Buchegger. Replica placement in p2p storage: Complexity and game theoretic analyses. In *ICDCS 2010, Proc.*, pages 599–609. IEEE Computer Society, 2010.

[32] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the pangaea wide-area file system. *SIGOPS Oper. Syst. Rev.*, 36(SI):15–30, Dec. 2002.

[33] R. Sharma, A. Datta, M. Dell Amico, and P. Michiardi. An empirical study of availability in friend-to-friend storage systems. In *P2P*, August 2011.

[34] P. Skowron and K. Rzadca. Exploring heterogeneity of unreliable machines for p2p backup. *CoRR*, abs/1212.0427, 2013.

[35] R. G. Tinedo, M. S. Artigas, and P. G. Lpez. Analysis of data availability in f2f storage systems: When correlations matter. In *P2P*, pages 225–236, 2012.

[36] L. Toka, M. Dell Amico, and P. Michiardi. Online data backup : a peer-assisted approach. In *P2P*, August 2010.

[37] L. Toka, M. Dell Amico, and P. Michiardi. Data transfer scheduling for p2p storage. In *P2P*, August 2011.

[38] D. N. Tran, F. Chiang, and J. Li. Friendstore: Cooperative online backup using trusted nodes. In *SocialNet*, 2008.

[39] C. Walter. Kryder's Law. *Scientific American*, 293:32–33, 2005.

[40] Y. Xing, Z. Li, and Y. Dai. Peerdedupe: Insights into the peer-assisted sampling deduplication. In *P2P*, pages 1–10, 2010.

[41] Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin. Bitvault: a highly reliable distributed data retention platform. *SIGOPS Oper. Syst. Rev.*, 41:27–36, April 2007.

[42] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST, Proc.*, pages 18:1–18:14, 2008.