2012 IEEE Fifth International Conference on Cloud Computing

F2BOX: Cloudifying F2F Storage Systems with High Availability Correlation

Raúl Gracia-Tinedo, Marc Sánchez-Artigas and Pedro García-López
Universitat Rovira i Virgili, Tarragona (Spain)
{raul.gracia|marc.sanchez|pedro.garcia}@urv.cat

***Abstract*—The increasing popularity of Cloud storage services is leading end-users to store their digital lives (including photos, videos, work documents, etc.) in the Cloud. However, many users are still reluctant to move their data to the Cloud due to the amount of control ceded to Cloud vendors. To let users retain the control over their data, *Friend-to-Friend* (F2F) storage systems have been presented in the literature as a promising alternative.**

However, as we show in this paper, *pure* F2F storage systems present a poor QoS, mainly due to availability correlations, which results in a loss of attractiveness by end users. To overcome this limitation, we propose a *hybrid* architecture that combines F2F storage systems and the availability of Cloud storage services to let users infer the right balance between user control and quality of service. This architecture, we called it F2BOX, is able to deliver such a balance thanks to the development of a new suite of data transfer scheduling strategies and a new redundancy calculation algorithm. The main feature of this algorithm is that allow users to adjust the amount of redundancy according to the availability patterns exhibited by friends. Our simulation and experimental results (in Amazon S3) demonstrate the high benefits experienced by end users as a result of the "cloudification" of F2F systems.

***Keywords*-friend-to-friend (F2F), storage systems, data availability, data transfer scheduling, availability correlations**

I. INTRODUCTION

In the last few years the Cloud Computing market has grown tremendously and new services are emerging steadily. One of these services is Cloud storage, which has rapidly developed to meet the increasing demand for people to handle their digital lives, including photos, media files, and work documents. The number of Cloud offerings is large and growing, ranging from Dropbox and the likes, to IaaS providers like Amazon.

Although Cloud systems can meet the requirements of mass storage, some users are reluctant to move data to the Cloud due to the large amount of control ceded to Cloud vendors, which includes the lack of data privacy [1] among other issues.

In this sense, social storage, also known as *Friend-to-Friend* (F2F) storage, has emerged as an attractive alternative to Cloud storage systems, as it places data control in the hands of users by which it significantly improves privacy and security [2], [3]. F2F storage also includes commercial storage applications like CrashPlan[1], which allow to back up data to friends.

In pure F2F storage systems, users have the choice to store their private information only on nodes they trust, and as such, they are *decentralized* in nature. However, their decentralized operation poses unique challenges that are actually hindering their adoption. The main one is the poor QoS offered to users compared with Cloud storage services, both in terms of *data availability* and *transfer times*. While Cloud storage services can offer multiple nines of data availability, a similar level of data availability may require a high amount of redundancy to be added to friends to mask their *discontinuous* participation. And even, it may be impossible to ensure it due to *availability correlations*. Indeed, measurements of online social networks have shown that friends present significant correlation in their activity patterns [4], [5]. This implies that it is not improbable to find all friends of a user simultaneously offline, for instance, during night hours, which makes it impossible to maintain high data availability even when placing one replica at each friend.

Moreover, such correlated patterns have an equally negative impact on transfer times. If a user were to upload one chunk of data to each of his friends, and most of them were offline due to correlation, the user would have to wait for those friends to come back online before completing all the data transfers.

These negative effects are even aggravated when the number of trustable friends is *reduced*, which is very common in social storage applications. To wit, over 63% of Facebook users have less than 100 friends [6], and what is even worse, most of their interactions occur only across a reduced subset of their social links. Concretely, 20% of their friends account for 70% of all interactions [6].

Our first contribution in this work is to analyze in depth the feasibility and performance of a F2F system in this scenario, a topic which has remained unexplored in the literature and is key to understand if F2F systems are really a valid alternative. Specifically, we analyzed the two fundamental QoS aspects of a F2F storage service: *data availability* and *transfer times*.

The analysis confirmed our intuition that it is *extremely hard for a F2F system to provide a good storage service* to users due to availability correlations. To improve the QoS, our idea was to take advantage of the superior availability of the Cloud to find the *right* balance between user control and QoS. In fact, Social Cloud is an emerging trend in the Cloud landscape [7].

The second contribution of this work is a hybrid architecture called F2BOX that combines F2F systems and Cloud storage to "get the best of both worlds". The design of F2BOX allows users to decide the level of QoS they want to obtain in basis to two parameters: *the targeted level of data availability* and *the fraction of data to be permanently stored in the Cloud*. At one end lies a user who wants a high QoS, for which the amount of data to be stored at the Cloud is high. At the other end is a user who wants high control over his data, for which he keeps most of his data stored at friends, but at the cost of a lower storage service quality.

To improve upload and download transfer times, F2BOX is accompanied by new scheduling policies at two levels: at the *friend-to-friend* level and at the *friend-to-Cloud* level. Further, a new method to adjust the amount of redundancy as a function of the availability patterns is introduced in F2BOX.

Finally, we analyze the existing trade-off between QoS and monetary cost. We evaluate our findings by simulation but also using a real prototype of F2BOX in our university campus. The prototype uses Amazon S3 as a Cloud storage service.

[1] http://www.crahsplan.com

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.22

The rest of the paper is structured as follows. We discuss the related work in Section II, and describe preliminary concepts in Section III. We study the performance of pure F2F systems in Section IV. In Section V, we present F2BOX, followed by its evaluation in Section VI. We conclude in Section VII.

II. Related Work

F2F storage systems originally emerged to overcome many of the limitations of P2P storage systems such as the *instability* and *heterogeneity* of peers [8], *free-riding* and the *lack of trust*. In fact, in the pioneering work of J. Li et al. [9], the authors argued that a user should choose its neighbors (the nodes with which it shares data) based on existing *social links* instead of *randomly*. Such an approach provided incentives for users to cooperate and resulted in a more stable system which, in turn, reduced the cost of maintaining data.

Friendstore [2] looked at how to ensure *availability* and *durability* in a social backup system by storing data on trusted nodes only, and hence discouraging free-riding. However, Friendstore evaluation was conducted by using availability traces of corporate desktop machines, which did not account for *availability correlations* existing in Internet systems.

Availability correlation is a well-known issue in large-scale distributed systems [10], [11]. Its impact on data availability has been extensively studied, especially in the P2P literature [12]. For instance, [13] argues that the average user availability may be misleading when it is used to calculate data redundancy in the presence of availability correlations. Also, authors in [14] introduced new metrics to quantify the degree of online-offline correlation of nodes. They applied these metrics to improve the performance of task scheduling and file storage applications.

The only work we are aware of that studied data availability in F2F storage systems is [4], which showed that F2F systems cannot guarantee high data availability: if no friends are online, then the data stored in the system will not be accessible by any means.

Similarly, the first research work investigating the problem of data transfer scheduling has been recently proposed in [15]. The authors provide a solid problem formalization as well as a novel approach to approximate the optimal time to schedule. They designed a set of data transfer scheduling algorithms for studying their performance in P2P backup applications.

Finally, another related area of research that overlaps with our work is *peer-assisted* storage that combines non-dedicated and Cloud resources into a single system.

For example, in [16], authors present a hybrid architecture where resources at peers (spare bandwidth, storage space) are complemented with temporal usage of Cloud storage services. They demonstrated that hybrid systems can be comparable to traditional client-server architectures but at a fraction of their costs. In this line, FS2You [17] is a large-scale online storage system with peer-based assistance and semi-persistent file availability that was developed to reduce server bandwidth costs. Their measurement study demonstrated the feasibility of combining peers with Cloud resources.

In summary, our work is the first to study the storage service performance of F2F systems, both in terms of data availability and transfer times, paying special attention on the effect of availability correlations and reduced friend-sets. Furthermore, we argue about the great importance of using a Cloud storage backend to improve QoS, and propose hybrid mechanisms to overcome the limitations of pure F2F systems.

III. Background

A. System Model

For a node v, we denote by $\mathcal{F}_v$ the set of friends at which v wants to store data. We assume this set is built up by leveraging upon real-trust between users, for example, in an online social network (OSN) like *Facebook* or *Orkut*. Since our focus is on domestic users, we assume that node n has *limited* download and upload bandwidth, denoted by d_v and μ_v, respectively. We also *limit* the number of parallel connections to P_d and to P_μ for downloads and uploads, respectively. The storage capacity at node v is denoted by s_v.

In addition, we assume that friends alternate between *online* (ON) and *offline* (OFF) states, with their online sessions being correlated over time. Correlation can be understood as the high probability that given an online user, his friends are online as well, which corresponds well with the strong diurnal pattern empirically observed in OSNs like *Facebook* [18].

To capture availability correlations, we distinguish between availability correlation for online sessions and correlations for offline sessions. As we will see next, this separation provisions us with valuable information about the impact of correlations on scheduling decisions that otherwise would remain hidden. This issue has not been well addressed in the literature.

Technically, to represent the availability of a host v, we use a vector AV_v of size T, where its ith position $AV_v[i] = 1$ if v was ON at time t_i, or -1 otherwise, where $t_i = \Delta \cdot i$, $\Delta > 0$. Δ represents the length of a time slot. With this representation, we can measure both types of correlations adapting the metrics in [14] into Definition 1:

Definition 1. The *Presence Matching* (PM) metric measures the level of coincidence of the ON sessions of two nodes a, b:

$$PM_{a,b} = \frac{|AV_a[t] \cap AV_b[t]|}{|AV_a[t] \cup AV_b[t]|}, \forall t \in T \text{ where } AV_i[t] = 1, i \in \{a, b\}\,.$$

Analogously, the *Disconnection Matching* (DM) metric gives the same information about OFF durations ($AV_i[t] = -1$).

We extend the above two metrics by calculating the average pairwise PM and DM measures within a group $\mathcal{F}$. We refer to these metrics by *Group Presence Matching* (GPM) and *Group Disconnection Matching* (GDM), respectively. Given a group $\mathcal{F}$, we calculate the GPM over $\mathcal{F}$ as follows:

$$GPM(\mathcal{F}) = \frac{\sum_{i,j \in \mathcal{F}, i \neq j} PM_{i,j}}{\sum_{i=1}^{|\mathcal{F}|-1} i}. \quad (1)$$

The calculation of GDM is analogous and has been omitted due to space constraints.

B. Data Availability

As the number friends is small and they can be temporarily offline, F2F storage systems provide data availability by means of redundancy.

To assure a given level of availability, our F2F system makes use of *Erasure Codes* (ECs), which has been proven to be more efficient in terms of redundancy than replication [19]. An EC scheme splits an input file into k fragments of $1/k$th the size of the original file. Then, these k fragments are encoded into n redundant blocks k, $k \leq n$, which are stored at different nodes to mask failures. The *data redundancy* required to store a file

is thus $\frac{n}{k}$. The original file can then be recovered by collecting any subset of k blocks out of the total n.

Ideally, the amount of redundancy used should be chosen to optimize transfer times provided that the target level of data availability is met. The reason is that the higher the amount of redundancy, the higher the quantity of data to be transferred to friends, and hence, the longer the scheduling times.

Traditionally, given the number of fragments for a file k and the target level of data availability A, the number of encoded blocks to upload n and hence, the redundancy rate $\frac{n}{k}$, has been determined as follows:

$$n = \min\left\{ x \in \mathbb{N} \,\middle|\, \left(\sum_{i=k}^{x} \binom{x}{i} \bar{a}^i (1-\bar{a})^{x-i} \right) \geq A \right\}, \qquad (2)$$

where $\bar{a}$ is the average host availability. (2) simply accounts for all the possible combinations of finding k blocks (or more) out of n, times the probability that this happens.

Two important observations must be discussed here about (2). The first is that this equation assumes that each fragment is stored at a distinct machine, because otherwise the failure of a single host would imply the loss of multiple fragments, thereby leading to an underestimation of the real data availability given by (2). This assumption is not realistic in our case. Due to the reduced number of friends (typically, between 5 and 20), it is very likely that a friend gets assigned more than one fragment.

Second, (2) assumes that hosts are not correlated, which is by far not true in F2F systems. As before, this implies that (2) can highly underestimate the real data availability. Considering a diurnal pattern in host availability, it is almost impossible for a reduced number of storage nodes, irrespective of the amount of redundancy used, to provide high data availability, as all of them will be probably offline during night hours.

For the above reasons, we have found it necessary to invent *a new algorithm to calculate the amount of redundancy to be added as a function of the level of availability correlation*. We are not aware of any similar treatment to this problem.

C. The Problem of Scheduling with Availability Correlation

Before going any further, it should be clearly stated what is meant by the term *scheduling*. We provide some definitions to clarify this concept.

Concretely, when we refer to a *transfer*, we mean the connection with a remote node that causes the transfer of a single block of data to it. Clearly, a transfer may be interrupted if the remote node becomes offline during this process. This takes an amount of time, namely, *block transfer time* (BTT). For this reason, we refer to a *schedule* as the set of transfers concerning the same data object.

Furthermore, we refer to as *scheduling policy*, the algorithm that decides the order according to which transfers must occur over time in order to minimize the time to complete a given schedule. We refer to the time to complete a schedule simply as the *time to schedule* (TTS).

Finally, we define two important concepts to understand the efficacy of scheduling policies further on in this article:

Definition 2. The *minimum time to schedule* (MTTS) is the time a node requires to transfer all blocks of a single schedule assuming ideal conditions, i.e., the MTTS only depends on the amount of data to transfer and the current bandwidth capacity.

Definition 3. The *optimal time to schedule (OTTS)* refers to the shortest TTS assuming dynamic participation from friends.

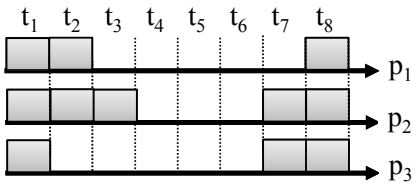


Fig. 1. Example of availability correlation.

To compute this time it is necessary to explore all the possible scheduling combinations with the *prior* knowledge of the exact ON/OFF pattern of each storage friend, which is not feasible in practice[2]. We use the OTT as a baseline in our evaluation.

Armed with the previous definitions, we are now in position to describe rather informally the problem of data scheduling on a small group of availability-correlated storage nodes by means of a simple formulation.

At this point, we assume that an appropriate (k, n)-erasure code has been selected according to the target data availability A and the availability correlations for the storage nodes in $\mathcal{F}$, with $|\mathcal{F}| \geq n$. Similar to [15], let $S(i,t)$ denote the event that an encoded block has been transferred to a friend i during time slot t. We say that a schedule S is *complete* if

$$\sum_{i=1}^{n} \sum_{t=1}^{TTS(S)} \mathbb{1}_{S(i,t)} = n, \qquad (3)$$

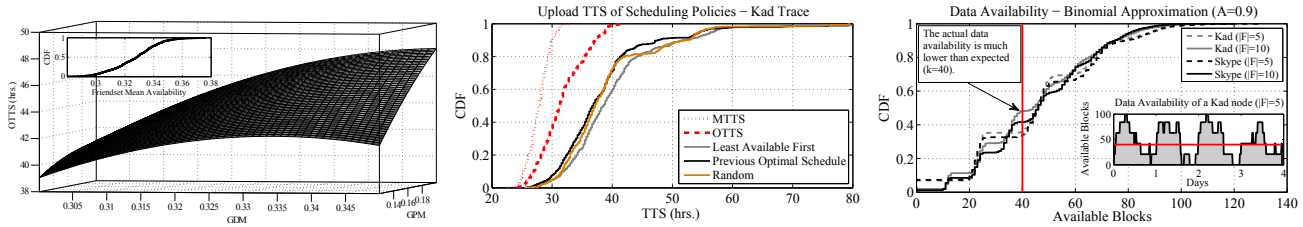
where $TTS(S)$ is the TTS for schedule S. Let $\mathcal{S}$ denote the set of all *complete schedules*. For simplicity, let us consider the set of schedules where each friend receives at most m fragments $\mathcal{S}_m$, i.e., $\mathcal{S}_m = \left\{ S \in \mathcal{S} | \sum_{t=1}^{TTS(S)} \mathbb{1}_{S(i,t)} \leq m, \forall i \in \mathcal{F} \right\}$. Then, the goal of the *scheduling policy* will be to find the schedule S in $\mathcal{S}_m$ with the shortest possible TTS.

Given a schedule S, note that its corresponding $TTS(S)$ is equal to $\max\left\{ t \in \mathbb{N}^+ | \mathbb{1}_{S(i,t)} = 1, i \in \mathcal{F} \right\}$. This implies that if the order in which transfers are to be executed do not take into account availability correlation, TTS may grow significantly if, for instance, the least available friend was scheduled last when all friends follow a diurnal pattern. This fact is mirrored in the example of Fig. 1, where $n = |\mathcal{F}| = 3$ and the set of potential schedules is S_1 (exactly one encoded fragment to each friend). We have depicted in gray the time slots where each node is online. This scenario highlights the importance of a good schedule when storage friends have correlated availabilities.

With an optimal schedule the owner would send a fragment to p_3 in the first time slot, then another to p_1 in time slot t_2, and finally one to p_2 in time slot t_3, concluding the schedule. However, if the first block is sent to p_2, then the second block to p_1, the owner will have to wait for p_3 to come online again in time slot t_8 to complete the upload. If the availability pattern of p_3 had been considered, this schedule would have been discarded.

In a F2F system, finding the *most optimal scheduling plan* is key to provide an efficient storage service in terms of transfer times, which is not trivial due to the *reduced number of friends* and *the availability correlations among them*. Indeed, a Cloud backend will be *fundamental* as shown in short.

[2]This time is computed by modeling the scheduling with dynamics friends as a *flow network*, and using binary search to find the *shortest-time max-flow* solution similar to [15].



(a) Impact of GDM and GPM on the upload *optimal time to schedule* (OTTS).

(b) Comparison of scheduling policies for a 1GB file and data availability $A = 0.9$ on *Kad* trace.

(c) Number of available blocks for 4-day time period for a 1GB file and data availability $A = 0.9$ on *Kad* and *Skype* traces.

Fig. 2. Storage service performance provided by a pure F2F storage system with $|\mathcal{F}| = 5$.

IV. QoS in F2F Systems: Friends are not Enough

In the section, we undertake an exploratory analysis to show that *pure F2F systems*, which have been coined as a potential alternative to commercial storage services such as Amazon S3, are not viable if availability correlations are high, for we need to resort to a hybrid architecture at the very least.

A. Impact of Availability Correlation on Transfer Times

First, we investigate the impact that availability correlation has on transfer times. For this analysis, we have synthetically generated 10K friend sets of cardinality 5 with different levels of GPM and GDM, and then we have calculated the *optimal time to schedule* (OTTS) for each one, in order to quantify the impact of correlation without side effects. The average online session for friends was of 8 hours while the average downtine was of 16 hours ($\bar{a} = 0.33$). Both ontime and offtime durations were drawn from Pareto distribution (heavy-tailed) with shape parameter $\alpha = 3$, which has been reported to provide a tight fit to the real lifetime distribution found in decentralized systems [20]. We uploaded a single file of 1GB size to each friendset by means of a $(40, 134)$–erasure code scheme. The uploaders had an upload rate μ of 30 kB/s. We place data blocks among friends in each friendset in a round-robin fashion.

In Fig. 2a, we depict the relationship between the degree of GPM and GDM and the resulting OTTS. Two key observations must be made.

Non-surprisingly, the first is that as the level of GDM grows, OTTS increases significantly due to the higher odds of *finding multiple friends simultaneously offline* (nocturnal patterns). If friends have low availabilities, even a moderate level of GDM could be catastrophic for the performance of a F2F system.

The other main observation is that, contrary to conventional wisdom, *a greater level of GPM may not be advantageous to the system and even undesirable* when the cardinality of friend sets is small. The fundamental reason is that many broadband users have asymmetric connections in which the upload rate is significantly lower than the download rate. Hence, if the GPM is high, while we are transferring a fragment to a friend, we are at the same time reducing the probability to find the remaining unassigned friends still online when that transfer is completed. This is readily apparent in Fig. 2a. For downloads, however, it must be noted that a moderate level of GPM is beneficial as it allows to exploit parallel connections.

B. Performance of F2F Scheduling Policies

As the *optimal time to schedule* (OTTS) cannot be achieved in practice for obvious reasons (it would imply the knowledge of the future ON/OFF pattern of users), here we examine the TTS of realistic scheduling policies and compared with OTTS. Even we propose a new scheduling policy. These policies are:

Random (RAN). This scheduling is computationally simple and very common in existing systems. It works as follows. In each timeslot, the user transfers one missing block to a storage node chosen uniformly at *random* among the available ones.

Least-Available First (LAF). This scheduling strategy is based on the assumption that nodes that have been online in the past will continue to do so in a near future. Therefore, it prioritizes transfers towards the nodes that have been less available within a past time window of w timeslots [15].

Previous Optimal Schedule (POS). We propose a novel policy which takes advantage from the *max-flow* based calculation of the *optimal time to schedule*. Essentially, it works as follows. If a scheduling were to be started at time t of day d, this policy would reproduce *exactly* the optimal schedule that would be obtained from the *max-flow* based approach but starting at time t of day d in the preceding week. This policy thus tries to take advantage of the *regular participation of nodes over the week*.

The setup for this experiment was basically the same as that described above but utilizing availabilities traces from *Kad* [10], a real P2P network. To instantiate the LAF policy, we set $w = 384$, which corresponded with a time window of 4 days with timeslots of $\Delta = 15$ minutes. The results of 1K schedules are shown in Fig. 2b.

As can be seen in the figure, all the three scheduling policies perform significantly worse than the *optimal schedule* (OTTS). Further, although POS performs *slightly* better than both RAN and LAF, there is not a "clear winner", which agrees very well with the results obtained in previous works [15].

Second, the difference between the OTTS and the MTTS is important. This result is very discouraging. In small correlated friendsets, it is extremely difficult for any scheduling policy to reach the *minimum time to schedule*, even in the case of using the optimal schedule with the *complete* knowledge of all the ON/OFF patterns of users.

Finally, it must be noted that the reason why the POS policy performs better than the other two, RAN and LAF, is because Kad nodes exhibit a quite regular participation over the week. In fact, the existing *autocorrelation* between the same day in consecutive weeks is *relatively* high. To gain better insight into this, we study the performance of the three policies depending on the mean level of availability autocorrelation for the nodes

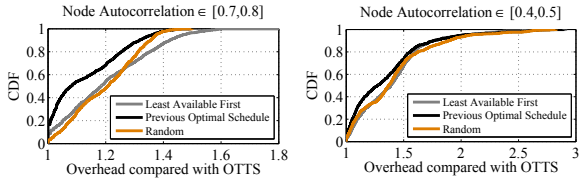


Fig. 3. Performance of scheduling policies as a function of autocorrelation.

in each friendset. The availability autocorrelation of a node v is simply the *autocorrelation coefficient* considering the vector AV_v as the input time series and selecting the appropriate *lag* in order to compare a week with its consecutive week. Recall that an output of 0 means *no correlation* while an output of 1 means *perfect correlation*. The results are plotted in Fig. 3

Observing this figure, we can conclude that the POS policy obtains important improvements over RAN and LAF when the autocorrelation of nodes is greater than 0.7.

C. Impact of Availability Correlation on Data Availability

As discussed in the Background section, the computation of the redundancy rate $\frac{n}{k}$ has been traditionally achieved through the binomial probability law (2), which overlooks availability correlations. In this section, we analyze what effect availability correlations have on the resulting data availability. To this aim, we conducted the following experiment.

In addition to *Kad*, we also used traces from *Skype* [11], and performed 1K schedules, uploading 1GB to each friendset. For each friendset, we used (2) to calculate the redundancy rate necessary to ensure a target data availability A of 0.9 (1 nine). We did it on a per-friendset basis to adapt the redundancy rate $\frac{n}{40}$ to the specific mean host availability of each friendset. Again, to preserve fairness, we spread the encoded fragments among the friends in each friendset in a round-robin fashion. Finally, for each friendset, we measured the resulting data availability for a time window of 4 days after the upload. More specifically, we recorded the number of available blocks in each timeslot, each timeslot had a duration $\Delta = 15$ mins, and aggregated all these values to obtain the CDF of the number of available fragments. We depicted the CDF in Fig. 2c.

Since we fixed $A = 0.9$, one should expect that 90% of the time, the number of available blocks was equal or greater than $k = 40$. However, as can be seen in Fig. 2c, nearly 50% of the time the number of available blocks is less than $k = 40$, which clearly shows the important overestimation in data availability provided by (2). The cause of this inaccuracy is the assumption that online sessions are not correlated, which translates into an *over-provisioning* of redundancy during diurnal hours and into an *under-provisioning* during night hours. In our opinion, such a treatment of redundancy is inadequate, and strongly hinders the viability of a pure F2F solution.

From this preliminary analysis, we conclude that due to the characteristics of F2F systems, it is extremely hard to provide an acceptable storage QoS to end users. Therefore, we propose to resort to the Cloud to provide realistic service guarantees.

V. F2Box: Cloudifying F2F Storage

A. System Design

To improve the performance of F2F storage, we propose *F2Box*, a hybrid architecture that utilizes the higher availability of Cloud storage services. As pure F2F systems, *F2Box* nodes use their social links to set up symmetric storage relationships among them. Furthermore, each *F2Box* node incorporates his *preferred* Cloud storage service as a storage node. Our hybrid model does not restrict the number of Cloud providers, thereby avoiding the *vendor lock-in* problem.

Cloud services improve the storage QoS of nodes in a two-fold manner:

1) When friends exhibit poor availabilities, Cloud storage is used to store a fraction of the data to assure the targeted data availability; and
2) As a *temporary buffer* to store blocks assigned to offline friends until they become online again. The idea of using the Cloud as a temporary buffer is to shorten the TTS by letting *F2Box* users to upload blocks to the Cloud instead of waiting for the disconnected friends to come back online again.

If the Cloud is used as a temporary buffer, then the friends themselves are responsible for downloading the missing blocks from the Cloud. In the end, the data owner removes the extra blocks from the Cloud upon a valid notification.

B. Providing Data Availability Guarantees

As discussed in the previous section, availability correlation in conjunction with a small friendset makes it hard to maintain a high data availability during the 24 hours of the day. For this reason, we strive to ensure a high data availability during the period of day where friends are mostly online, what we call as *daily data availability*. More specifically, our aim is to assure a high data availability during at least δ hours per day, instead of at all times as in traditional storage systems, where a sufficient number of uncorrelated nodes can be found like P2P systems.

To this end, we propose a novel hybrid redundancy scheme where a fraction of the data is permanently stored in the Cloud and the rest is maintained by the friendset. More specifically, we denote by f_{cloud} the fraction of the files to be *permanently* stored at the cloud. A high value of f_{cloud} translates into a high data availability, since any Cloud service has been designed to reach several nines of uptime availability. Notice that for a $F_C < 1$ there will be less than k blocks permanently stored in the Cloud. This preserves the distinctive *data privacy* feature of F2F systems, since the Cloud vendor *cannot reconstruct the original file* by any means.

The remaining proportion of the file $1 - f_{\text{cloud}}$ is maintained by the friendset. Given a chosen k, this requires a new method to calculate the minimum redundancy rate $\frac{n}{k}$ according to the specific availability patterns of friends, so that data availability can be maintained at least δ hours per day. Recall that a file is available if at least k blocks are available for download.

The algorithm for this computation works as follows. Given the number of blocks k, the algorithm computes the number of blocks to be uploaded to the Cloud n_{cloud} as $n_{\text{cloud}} = \lceil k f_{\text{cloud}} \rceil$. The initial number of blocks to be transferred to the friendset is $n_{\text{friend}} = \lfloor k(1 - f_{\text{cloud}}) \rfloor$. The algorithm then assigns the n_{friend} blocks to the friends in a *round-robin* style in order to balance storage costs. Using a past time window of w timeslots, it computes the number of timeslots $n_{timeslot}$ within the window where the number of available blocks $n_{\text{cloud}} + n_{avail}$ is equal or greater than k. Note that at least few days should be considered in the time window w. If $n_{timeslot}$ times the duration of a timeslot Δ covers δ hours, the algorithm halts, and the values n_{cloud} and n_{friend} are returned. Otherwise, n_{friend} is incremented by one block and the entire process is repeated. This procedure is repeated again and again until the value of

n_{friend} guarantees δ hours of data availability, whenever n_{friend} does not exceed $|\mathcal{F}_v|$ replicas. Then, the redundancy rate is simply $(n_{\text{cloud}} + n_{\text{friend}})/k$. The pseudocode for this algorithm is shown below.

Algorithm 1: Hybrid historical redundancy calculation

```
Input: F_v, δ, f_cloud, k
Output: n_cloud, n_friend
n_cloud ← ⌊k · f_cloud⌋;
n_friend ← ⌈k · (1 − f_cloud)⌉ − 1;
h_avail ← 0
while h_avail ≤ δ and (n_friend / k) < |F_v| do
    n_friend = n_friend + 1;
    n_timeslot ← 0;
    for i in w do
        n_avail ← 0;
        for u in F_v do
            if AV_u[i] ≡ 1 then
                n_avail = n_avail + n_friend/|F_v|;
            end
        end
        if n_cloud + n_avail ≥ k then
            n_timeslot = n_timeslot + 1;
        end
    end
    h_avail ← n_timeslot · Δ
end
```

Note that Algorithm 1 gives flexibility to the user regarding the amount of data he wishes to store in the Cloud. Moreover, depending on the targeted data availability δ, the monetary cost of Cloud storage will be higher or lower, and the disk capacity required to friends will be on the opposite situation. We study this trade-off in the next section.

C. Improving Scheduling Times

Given determined the redundancy rate in terms of n_{cloud} and n_{friend}, we resort to the Cloud in order to decrease scheduling times. As described above, the Cloud provider, in addition to store n_{cloud} blocks, acts as a temporary repository to store the blocks assigned to nodes that are currently offline. This policy can lead to an important reduction of scheduling times.

To this aim, we propose the *Bandwidth Maximizing Friend-to-Cloud* policy[3]. With this policy a user seeks to minimize scheduling times as much as possible by fully utilizing his own bandwidth. Thus, if a node responsible for a block is not online, this policy automatically pushes this block to the Cloud. This ensures that a node achieves the MTTS. In this sense, this scheduling policy can be viewed as a pure F2F scheduling policy such as *Least Available First* or *Previous Optimal Schedule* that immediately uploads a block to the Cloud when the corresponding friend to which transfer that block is offline.

In any case, the extra fragments allocated to the Cloud will have to be downloaded afterwards by the friends to whom were initially assigned. Clearly, this yields extra Cloud costs in outgoing traffic, which are analyzed in Section VI-B.

VI. Evaluation

A. Setup & Methodology

In this section, we describe the simulation environment used to assess the performance of F2Box.

[3]In this piece of research we designed other Friend-to-Cloud policies. However, the differences in performance among them were not significant.

Parameter Description and Values		
N	Nodes in the system	60 (Skype), 110 (Kad)
$\bar{a}$	Node mean availability	$[0.2, 0.6]$
T	Simulation time	12 days
μ, d	Node upload/download bandwidth	30KBps, 120KBps
s	Node storage capacity	500GBytes
P_μ, P_d	Parallel upload/download connections	1, 4
k	Original file fragments	40
β	Object size	2GB
$\|\mathcal{F}\|$	Friendset size	10

TABLE I
SYSTEM PARAMETERS AND DESCRIPTION.

In our tests, we modeled the alternating ON/OFF behavior of nodes using real availability traces. Since a capital aspect of our evaluation lies on studying the impact of availability correlation, we employed traces from *Kad* [10] and Skype [11]. From these traces, we excluded all the nodes whose availability was out of the range $[0.2, 0.6]$ during the simulation window of $T = 12$ days. From this subset, we filtered the nodes that were not online at least once every 48 hours. This filtering process was necessary to exclude from simulation extreme availability cases, such as superpeers or *permanent* churn.

Since we do not target backup scenarios, where each storage operation involves several Gigabytes of content, we consider that the data owner stays connected during the whole storage operation. This behavior is natural in file-sharing and storage applications. In addition, each friendset served a single storage request. The impact of concurrent schedules within a friendset was addressed in the experimental evaluation.

Our simulations are divided into three different phases, each during 4 days:

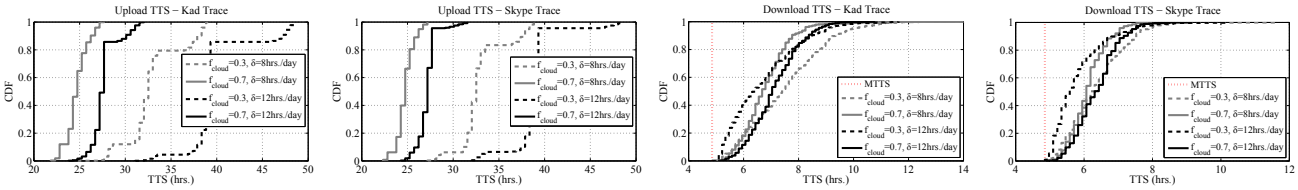
1) *Training phase*. In this phase, the users collect historical information in form of availability vector AV about each of their respective storage friends. This knowledge base is vital to calculate the value of n_{friend}, or to initialize the scheduling policies LAF and POS, among other matters.
2) *Upload phase*. During this phase, the data owner uploads a single file of β bytes, plus the associated redundancy, to the system.
3) *Download phase*. Finally, in the *download phase* a user retrieves k data blocks from the system to reconstruct the uploaded file.

All the results presented below were obtained by repeating this process for $1,000$ random friendsets in each configuration.

Our analysis showed that applying scheduling policies in this scenario has little or no effect on download TTS. For this reason, download schedules are performed randomly, giving a greater priority to the available blocks stored at friends in order to save costs due to data transferred out of the Cloud. Only in case of having idle connections, a node retrieves fragments from the Cloud.

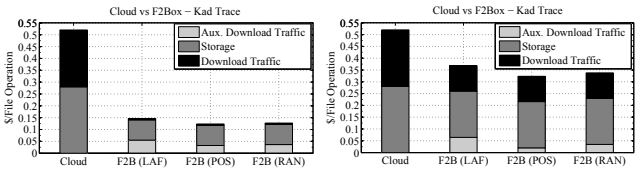
Monetary Cost. We adopt for our evaluation the same pricing model of Amazon S3, which is a well representative for Cloud storage. Accordingly, we consider outgoing data transfers and storage to be charged by the Cloud service. At February 2012, Amazon S3 pricing was of $0.120 per GB of data transfer out of the Cloud and $0.140 per GB/month of storage. Transfers into the Cloud are free of charge. Finally, it must be noted that we did not account for the cost incurred by transactions (GET, PUT, COPY, POST, and LIST requests) due to the small charge per transaction: $0.01 for every $10,000$ transactions.

We compare the costs of F2Box with the same service provided by Amazon S3. We analyze the costs by *file operation*,



(a) Upload TTS for Kad (b) Upload TTS for Skype (c) Download TTS for Kad (d) Download TTS for Skype

Fig. 4. Scheduling times as a function of f_{cloud} and the targeted data availability δ for friendsets of size $|\mathcal{F}| = 10$.



(a) $f_{\text{cloud}} = 0.3$. (b) $f_{\text{cloud}} = 0.7$.

Fig. 5. Comparison of monetary costs of schedules between Amazon S3 and F2Box for one month storage in Kad with $|\mathcal{F}| = 10$ and $\delta = 12$ hours/day.

TABLE II
DATA AVAILABILITY

$\delta = 8$ hours/day	Kad ($\|\mathcal{F}\| = 10$)			Skype ($\|\mathcal{F}\| = 10$)		
Percentile	*0.25*	*0.5*	*0.75*	*0.25*	*0.5*	*0.75*
$f_{\text{cloud}} = 0.3$	5.09	6.67	8.16	6.59	8.10	9.40
$f_{\text{cloud}} = 0.7$	4.76	6.49	7.97	6.52	7.96	9.37
$\delta = 12$ hours/day	Kad ($\|\mathcal{F}\| = 10$)			Skype ($\|\mathcal{F}\| = 10$)		
Percentile	*0.25*	*0.5*	*0.75*	*0.25*	*0.5*	*0.75*
$f_{\text{cloud}} = 0.3$	8.84	10.54	11.89	10.28	11.59	12.77
$f_{\text{cloud}} = 0.7$	8.88	10.52	11.89	10.29	11.55	12.77

which corresponds to upload and download a file of β bytes, as well as keeping it stored for one month in the system.

B. Simulation Results

Scheduling times. In first place, we comment on the scheduling times obtained by F2Box and depicted in Fig. 4a for Kad, and in Fig. 4b for Skype, respectively. As can be seen in both figures, the higher the amount of data transferred to the friends, the longer the upload *time to schedule*. This is because a higher value of $1 - f_{\text{cloud}}$ translates into a greater number of blocks n_{friend} to be uploaded in order to assure the same level of data availability. This fact is clearly visible in the case of demanding a *high data availability* like a δ of 12 hours/day.

It is worth to note that the steeper slope that is observed in Fig. 4a and 4b decreases as more blocks are stored at friends. The reason of this lies on the redundancy calculation we make to ensure data availability in the presence of high availability correlation. However, the availability traces we use in our tests are only *moderately* correlated, which very often results in an increase in the redundancy assigned to friends, so as *the time to schedule*. This is clearly visible in the "flattened" region of the curves for $f_{\text{cloud}} = 0.3$. Such an increase of redundancy is not *inversely linear* with the level of availability correlation, which is especially apparent in friendsets of small cardinality. In fact, we observed that this flattened region disappears completely in friendset of 30 or more members.

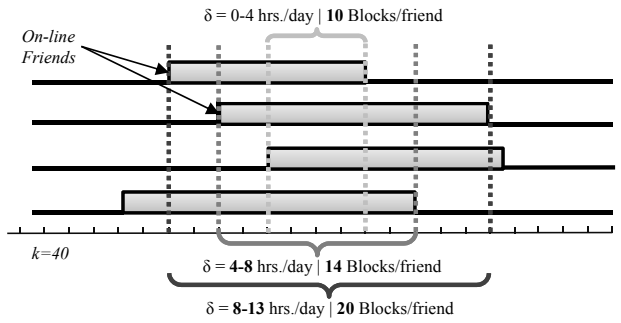


Fig. 6. Relationship between daily data availability (δ) and the amount of redundancy (blocks/friend). Clearly, this has a tremendous impact on the TTS.

We illustrate the relationship between *data availability* and *redundancy* at storage nodes in Fig. 6. By simple inspection of this figure, it is easy to see that a targeted daily data availability of 8 hours requires much lower redundancy than a little higher data availability of 9 hours.

The download scheduling times are close to the MTTS as shown in Fig. 4c and Fig. 4d. Even in the worst case, between 65% to 90% of downloads are completed in less than 8 hours when the MTTS is of $4,85$ hours. It is interesting to note that assigning more redundancy to friends contributes definitely to the smaller download TTS compared with upload times. On the other hand, the *relatively* long tail of the download TTS distribution, particularly for Kad, suggests that uploading more blocks to the Cloud significantly reduces the chances to exceed greatly the average download TTS. This was expected since a user can always resort to the Cloud when their friends are unavailable or present a high GDM.

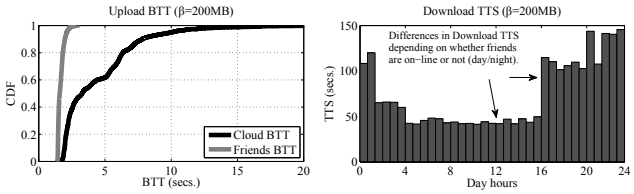
Data availability. In Table II, the data availability obtained by our system is reported for different values of f_{cloud}.

First, we observe that our redundancy calculation algorithm is very accurate in the presence of availability correlation. This means that past availability information is suitable to assure δ hours of data data availability for small, correlated friendsets. Proof of that is that the *median* data availability is quite close to the targeted data availability δ, particularly for Skype.

Further, we do not observe significant differences between the data availability obtained depending on the fraction of data stored in the Cloud (f_{cloud}). This means that our algorithms performs well irrespective of the storage distribution between friends and the Cloud.

Second, as f_{cloud} grows, the amount of data redundancy needed to achieve a certain δ is clearly lower, as can be inferred from the upload times (figures 4a and 4b). Therefore, by making use of Cloud storage, F2Box *enables equal or higher levels of data availability than a F2F system with much less data redundancy*. This is key to limit scheduling times and improve the storage capacity of a F2F system.

Cost-QoS trade-off of F2Box. First, it is clear that there exists a



(a) Upload BTT distribution of uploaded blocks (Cloud/friends).

(b) Impact of diurnal patterns on the Download TTS performance.

Fig. 7. Transfer performance of our hybrid F2F prototype.

trade-off between Cloud monetary costs and upload scheduling times as shown in Fig. 5a and Fig. 5b. More specifically, if a user prefers to minimize the time spent in storing large files, it is generally better uploading more data to the Cloud. This will reduce the number of blocks to be transferred to friends n_{friend}, thereby drastically reducing the upload time to schedule.

The *inverse* trade-off arises from downloads. Storing more data at friends reduces monetary costs because less fragments must be downloaded from the Cloud. Therefore, if some file is retrieved very often, it would pay off to decrease f_{cloud} and upload more data to friends.

Finally, it worth to mention that the specific F2F scheduling policy chosen as basis for *Bandwidth Maximizing Friend-to-Cloud* policy has little impact on monetary costs, so the use of the Cloud as a temporary buffer contributes minimally to the total cost. This cost is termed *auxiliary download bandwidth* in Fig. 5. However, it must be noted that POS outperforms LAF and RAN, mainly for two reasons: 1) The nodes in the traces are *autocorrelated* enough to exploit their previous behavior; and 2) the transfer of blocks into the Cloud *indirectly* increases the autocorrelation of the schedule, as those blocks will always be uploaded in the specified timeslot by POS.

In conclusion, *F2Box is flexible enough to cover all user needs by trading daily data availability and scheduling times for monetary cost*. This opens the door to a real adoption of F2F systems thanks to the greater reliability of Cloud storage.

C. Experimental Results

The implementation of the F2BOX system is called FRIENDBOX (http://ast-deim.urv.cat/friendbox/). We deployed a group of FRIENDBOX users in the Universitat Rovira i Virgili campus, which are friends among them in Facebook. In our scenario, users perform and serve storage operations in the system. Three of those nodes are *storage requesters*, that is, they are automated `FriendBox` nodes, whereas the rest are real users. In this section, we analyze the service received by requester nodes in a real `FriendBox` deployment. Real users are connected to the system depending on the time they spend at the campus —this introduces *availability correlations* in our experiments. Thus, requester nodes are continuously (24 hours) storing/retrieving a $\beta = 200$MB file and therefore they are always available. Requester nodes are configured with $f_{\text{cloud}} = 0.5$ and $n/k \approx 2.5$. Moreover, FRIENDBOX uses Amazon S3 as a Cloud provider.

Firstly, we observe in Fig. 7a how the vast majority of block transfers among FRIENDBOX nodes are faster than resorting to the Cloud. This is mainly caused due to the high speed network available in the campus (Fast Ethernet, 100Mbps).

Second, one of the most interesting points refers to the download TTS performance depending on the moment of the day. We can observe that scheduling times are significantly shorter between $[4\text{h}-16\text{h}]$ than in the rest of the day. Actually, this is the time range where the majority of friends are online. From this result, we conclude that in this kind of scenario (university, business) the availability of friends importantly improves the TTS performance.

VII. CONCLUSIONS

In this paper, we have seen why *pure* F2F storage systems cannot provide a storage service comparable to Cloud offerings due to availability correlations. To retain the main advantages of these systems, we have proposed a *hybrid* architecture that takes advantage of the superior availability of Cloud storage services to improve QoS. To this aim, we have developed novel scheduling strategies, and a new algorithm that let users adjust redundancy according to the *availability correlation* exhibited by friends. Both simulation and experimental results certify the benefits of combining the best of both worlds.

ACKNOWLEDGEMENTS

This work has been partly funded by the Spanish Ministry of Science and Innovation through projects DELFIN (TIN-2010-20140-C03-03) and RealCloud (IPT-2011-1232-430000).

REFERENCES

[1] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Software Engineering Challenges of Cloud Computing*, 2009, pp. 44–52.
[2] D. N. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *SocialNets'08*, 2008, pp. 37–42.
[3] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social cloud: Cloud computing in social networks," in *IEEE CLOUD'10*, 2010, pp. 99–106.
[4] R. Sharma, A. Datta, M. Dell'Amico, and P. Michiardi, "An empirical study of availability in friend-to-friend storage systems," in *IEEE P2P'11*, 2011, pp. 348–351.
[5] S. A. Golder, D. M. Wilkinson, and B. A. Huberman, "Rhythms of social interaction: Messaging within a massive online network," in *Communities and Technologies*, 2007, pp. 41–66.
[6] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *EuroSys'09*, 2009, pp. 205–218.
[7] G. Pallis, "Cloud computing: The new frontier of internet computing," *IEEE Internet Computing*, vol. 14, no. 5, pp. 70–73, 2010.
[8] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *HotOS'03*, 2003, pp. 1–6.
[9] J. Li and F. Dabek, "F2f: Reliable storage in open networks," in *IPTPS'06*, 2006.
[10] M. Steiner, T. En-Najjary, and E. Biersack, "A global view of kad," in *ACM SIGCOMM IMC'07*, 2007, pp. 117–122.
[11] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer voip system," in *IPTPS'06*, 2006.
[12] M. Sánchez-Artigas and E. Férnandez-Casado, "Evaluation of p2p systems under different churn models: Why we should bother," in *Euro-Par Parallel Processing*, 2011, pp. 541–553.
[13] R. Dunn, J. Zahorjan, S. Gribble, and H. Levy, "Presence-based availability and p2p systems," in *IEEE P2P'05*, 2005, pp. 209–216.
[14] S. Le Blond, F. Le Fessant, and E. Le Merrer, "Finding good partners in availability-aware p2p networks," in *Stabilization, Safety, and Security of Distributed Systems*, 2009, vol. 5873, pp. 472–484.
[15] L. Toka, M. Dell'Amico, and P. Michiardi, "Data transfer scheduling for p2p storage," in *IEEE P2P'11*, 2011, pp. 132–141.
[16] ——, "Online data backup: A peer-assisted approach," in *Peer-to-Peer Computing (P2P)*, 2010, pp. 1–10.
[17] Y. Sun, F. Liu, B. Li, B. Li, and X. Zhang, "Fs2you: Peer-assisted semi-persistent online storage at a large scale," in *INFOCOM*, 2009, pp. 873–881.
[18] S. A. Golder, D. M. Wilkinson, and B. A. Huberman, "Rhythms of social interaction: Messaging within a massive online network," in *Communities and Technologies*, 2007, pp. 41–66.
[19] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *IEEE P2P'04*, 2004, pp. 90–97.
[20] Z. Yao, D. Leonard, X. Wang, and D. Loguinov, "Modeling heterogeneous user churn and local resilience of unstructured p2p networks," in *ICNP'06*, 2006, pp. 32–41.