Eurecom
Department of Networking and Security
2229, route des Crêtes, B.P. 193
06904 Sophia-Antipolis, FRANCE

Research Report RR-10-236

On Peer-assisted On-line Backup System Design Choices

Laszlo Toka, Matteo Dell'Amico, Pietro Michiardi

Tel : (+33) 4 93 00 81 45
Fax : (+33) 4 93 00 82 00
Email : {Pietro.Michiardi}@eurecom.fr

[1]Eurecom's research is partially supported by its industrial members: BMW Group Research & Technology - BMW Group Company, Bouygues Télécom, Cisco Systems, France Télécom, Hitachi Europe, Orange, SFR, Sharp, ST Microelectronics, Swisscom, Thales.

Abstract

This paper presents our work on online data backup systems in which users store their backup data in data centers and/or on peers. We analyze the system performance of various data placement schemes and evaluate the choices of implemented transfer scheduling techniques with different user resource profiles.

Keywords: peer-assisted, peer-to-peer, backup, system design

1 Introduction

1.1 Understanding On-line Backup Systems

Our work aims to study peer-assisted online backup service solutions that offer safe storage of personal data for Internet subscribers. Already existing backup solutions are numerous, e.g., on-site, online to data center, online to cloud backup. We suggest a hybrid approach that is different from existing schemes: we start from an online data center scheme and we build a peer-assisted system in order to offload the data center (i.e., storage & bandwidth burden) using idle peer resources (storage & bandwidth of peers) with the goal of decreasing costs of bandwidth and storage. This is achieved by "outsourcing" backup data on peers for storage, thereby minimizing data center costs while sustaining the quality of the backup service.

1.2 Our goal — Primary and secondary objectives

We study the impact of different centralized data-placement algorithms in the following three possible approaches for online backup, with the focus on the placement per se.

- data center-based (DC) online backup;
- peer-assisted (PA) online backup: partition of backup data load between data center and peers;
- peer-to-peer (PP) online backup.

To attain our goal, we perform numerical simulations since complex issues can easily be taken into account with a simulator while a simple theoretical model (e.g. calculation of the available aggregate space on peers) cannot consider the effects of interrupted data transfers (e.g. when peers go offline), upload bandwidth constraints, etc. We show in our simulations that in many cases either upload bandwidth or storage is the bottleneck, while the simple existing models only consider peer availability as the limiting factor in a PP backup system. Note, however, that if a peer's bandwidth is limited, the data it can backup is limited even if it provides unlimited disk space for others and conversely, if disk space is limited, high bandwidth does not give any remedies.

Our secondary objective is, once we see the different data placements' performances (along with other crucial system design choices, e.g., data transfer scheduling), we design distributed peer selection algorithms accordingly in order to realize the system that offers the best trade-off on quality of service and costs. Without jumping into the details, the following peer selection algorithms will be studied:

- random peer selection (offers a DHT-like [1] solution);
- clustering (selective) peer selection based on peer average availabilities ([5]).

The reason for clustering is that it inherently embraces an incentive mechanism and effectively copes with real-life online patterns of the users.

1.3 Our assumptions

Here we state our assumptions, that give reasonable considerations about peer resources. We build our analysis on these assumptions.

- Excess storage and bandwidth capacity of peers is free (i.e., if the resource is not used it is lost), moreover storing backup on remote peers is not billed for the user.

- Over-provisioned data center possesses unlimited storage capacity and bandwidth (i.e., the bottleneck is always the peers' uplink/downlink).
- Long-term storage on data center costs more than the data center's bandwidth utilization (based on the pricing of Amazon S3 [4], uploading and downloading data once costs roughly two months of storage).
- There is no correlation among a peer's characteristics (that are supposed to be fixed in time): online behavior, link capacity, storage capacity are a peer's resources and their amounts are independent one of another, and the storage requirement is its demand which is independent of all of its resources.
- Each peer keeps a copy of its own backup locally (without redundancy), therefore the backup data is available except in the case of disk failure, when all the local data is lost.
- We target the design of a backup system, therefore downloading data from data center or remote peers is only required when the locally stored copy is lost: downloading backed up data happens only for restoring purposes.
- Peers maintain their own data, i.e., they reconstruct all their backed up data in the system in case of a remote peer's death or disk failure. Note, that peers are able to re-generate encoded backup if one or more of their remote peers lose(s) data without downloading the backup to restore before encoding.

1.4 Metrics of evaluation

We investigate the following metrics closely, in order to quantify the performance of various system design choices. We show the results of our simulations in the following metric perspectives:

- Time-to-backup (TTB): *the time spent from the start of uploading a peer's data to the completion time of uploading sufficient amount of data in order to achieve the targeted quality of backup (the target quality level is defined later)*
- Time-to-restore (TTR): *after a peer has backed up* sufficient *amount of data, it is the time required to restore locally its whole original data starting at a given point in time*
- Monetary costs [$]: [1]
  - Storage cost of data center per peer[2]
  - Traffic cost of data center per peer[3]
- Peer and network provider costs:
  - Storage and bandwidth load on peers (due to backup, restore, repair)
  - Aggregate traffic cost of network provider (due to backup, restore, repair) [4]

2 Related work

Existing work does not solve the problem we are targeting for several reasons: on one hand storage systems are more often envisaged than *backup* systems, therefore data availability requirement receives particular importance and cannot be always fulfilled [6]; on the other hand in related works about P2P solutions the focus is on data redundancy schemes and data repair techniques, while little work has been done on hybrid

[1] In DC, users spend e.g., 100$, but if they obtain the same service quality exploiting surplus storage and bandwidth on peers which is free, PA only costs e.g., 40$. Thus, cost saving is equivalent with evaluating how much data the whole system can store on the decentralized part.

[2] The costs are calculated based on the pricing of Amazon S3 [4]: 1-month storage and downloading 1 GB data costs 0.15$.

[3] Bandwidth cost is a direct function of the "load" on the data center.

[4] Maintenance: in clustered systems internal churn occurs (i.e., re-establishing the clusters of peers) which is triggered by change in peers' availabilities. However, imposed incentives probably do not cause varying behavior for peers, but rather a different choice of behavior to start with.

approaches to mitigate the shortcomings of P2P systems: to the best of our knowledge, no prior work tackles peer-assisted online backup systems.

Furthermore, while peer selection in related works is dictated by well-known data indexing-retrieval techniques (e.g., DHT, locality-based), hence random peer selection; our scheme is based on shared resources. Since scheduling, redundancy coding rates and repair techniques are not orthogonal to our focus on peer selection, we provide discussion on those aspects as well. The following topics however fall outside the scope of our work, and are based on state-of-the-art techniques:

- Peer monitoring: heartbeat based, central server collects measurements;
- Data encoding: we use basic erasure coding techniques.

3 Important components of a backup system

Here we discuss in details the main contribution of our work on the design of a PA or PP system. These systems require careful control of quality of service, and the system scale, the peer heterogeneity in their service demand, resources and contribution must be taken into account. The problems that we focus on and for which we suggest different techniques are:

- data placement;
- data structure;
- data transfer scheduling;
- data repair.

3.1 The data-placement problem

Data placement problem is about deciding where a peer will store parts of its backup data. Note, that this is a non-trivial operation that deals with the decision to store data on: servers only (DC), servers and peers (PA), peers only (PP). Additionally, whenever remote peers are selected, we explore different cases of selection criteria, symmetric and asymmetric data exchange options.

Outsourcing as much storage load as possible to peers from data center has the advantages of cost reduction in terms of storage and bandwidth of the central component, exploiting excess resource capacities at the peers (distributed bandwidth consumption, storage space, online uptime availability). Storing on remote peers also improves "reliability", since private backup data are not in the hands of a single company that may go out from business. Our analysis on data placement answers the questions about the data amount that can fit in a PA system on the data center and on the peers respectively, and about the impact of data placement on this amount. Results are shown for different parameters and scenarios later in the paper.

In many well known P2P applications (e.g., file sharing), resources are scattered irregularly between peers and there is need for creating an indexing/search infrastructure (e.g., a DHT) to efficiently locate the data. Furthermore, most existing P2P backup/storage solutions spread data randomly between the peers in the network, and mimic this solution in order to become able to recover the data location. We take a different path: we recognize the possibility of choosing *where* to store data. We can exploit this degree of freedom to optimize data placement of our system with respect to resources (i.e., bandwidth, storage). Therefore we investigate if a stratified system performs better than any other approach to peer selection, and we show the advantages and limitations of a stratified system on the spectrum of different peer contributions. Moreover, we study the outcome difference between symmetry or asymmetry in peer selection (symmetric contribution: a peer stores data on a remote peer, the latter also stores the same amount of data on the first peer) and we arrive at the conclusion that answering this question requires knowledge about the system scale and the heterogeneity of peers consisting the system.

Three different techniques of data placement are analyzed (assuming centralized control):

1. All data stored centrally (DC). The data center bears all storage and the implied traffic burden, moreover the cost of online availability. The backup data transferred to the data center should not be redundant coded to save on traffic cost.

2. Part of data stored centrally and part of data stored on peers (PA). This case describes a system where the data center outsources as much storage load to peers as possible in order to offload the data center, while guaranteeing the service level requirements (e.g. TTR). Thus, peers "assist" to the data center. In terms of traffic, this scheme has the advantage of "caching" a part of the backup data at peers [2], from where retrieving data is free. Central storing capacity is mandatory

   (a) if the overall storing capacity at peers is not sufficient to store the data with the necessary redundancy for every user;

   (b) if there is not enough available peer present in the system that is suitable to store data.

   If these constraints cease to exist due to additional peer storage capacity or late arrival respectively, storage relations can be built between peers (that store data on the server) by re-transferring backup data among peers to offload the data center in storage and traffic. If the peers' storage capacities do not cause limit, the load on peers might be distributed

   (a) uniformly: peers store the same amount from the overall backup data, therefore peers contributing more uptime and larger bandwidth will suffer.

   (b) inversely proportionally to their uptime contribution: this mitigates the oppressing effects of the previous design: peers can trade their uptime resources for storage space and traffic.

3. All data stored on peers (PP). The data center is fully offloaded, on the other hand, the network provider experiences higher traffic cost due to the additional data redundancy, which becomes necessary when storing solely on unreliable peers. If the aforementioned constraints exist, for some peers the quality of service might not be acceptable.

3.1.1 Data placement on peers - The Tracker

In PA and PP systems, each peer constructs its neighborhood by querying a tracker to select and collect remote peers for storage starting from its first appearance. At startup, a peer contacts the tracker[5] and asks for a peer set of fixed size. The tracker can reply with

- a randomly selected subset of peers (randomized peer selection);
  - DHT-like or unstructured network
  - Pros: Easy to implement, easy to distribute, load balancing
  - Cons: load balancing, does not tackle heterogeneity
- a randomly selected subset of peers with the **same** peer average availability parameter (clustered peer selection).
  - Clustered with respect to availability: store only on peers that have similar availability and sufficient bandwidth characteristics since data availability is directly defined by the availability of peers storing it. Stratification of peers leads to less storage and traffic burden on highly available peers, thus creates incentives.
  - Clusters are defined by boolean criteria, i.e., either a node belongs to a cluster or not. If the size of a cluster is too small to store data, all the excess data is stored in the data center. Data placement is essentially randomized peer selection within the peer's cluster.
  - Pros: tackles heterogeneity, incentive
  - Cons: measuring/calculating peer availability, internal churn, granularity of the clusters' peer characteristics (clustering within a cluster may exclude peers from storage relations)

[5] In the simulation a global array mimics the tracker. This does not impact the outcome of the simulation: if a suitable peer is present in the system, it gets chosen.

The tracker does not check whether the returned peers are online at the request time; those peers can be contacted that are online when the selecting peer is, and still have sufficient unallocated storage capacity to back up a part of the selecting peer's data. In case the number of peers provided by the tracker is not sufficient to store all data fragments, and/or if not enough of the known peers are available, a peer can request another fixed number of peers. The backup operation on peers temporarily fails if there are not enough peer in the system or not enough peers correspond to the selection criteria.

Imposing symmetric (bilateral) storage relation is an option. Note, that the maintenance operation (i.e., restoring lost fragments due to disk crash or peer death) may require the peer set to be re-constructed.

3.1.2 Storing on data center

The data placement problem in PA systems consists of the decision about when and how much data will be stored on the data center. We study the possible alternatives listed below:

- *Opportunistic*: data is transferred to data center with excess upload capacity, and it is continuously deleted as the amount of data that is successfully stored on peers is growing in the meantime;
- *Pessimistic*: the necessary amount of data is uploaded to data center with full upload capacity in order to be able to restore backup, then uploads start toward remote peers, and centrally stored data is continuously deleted as the amount of data that is successfully stored on peers is growing.

3.2 The data structure problem

The data structure problem tackles data fractioning and applied redundancy rate when delegating storage responsibility to peers. We show that the number of peers and the quality of peer resources and contributions highly affect the quality of service (TTR) that can be improved by choosing well-suited data structure. In related work, creating data redundancy usually performed by data replication or erasure coding [7], and determining redundancy rate is based on a calculation by supposing that data parts are stored on different remote peers with homogeneous average availability and independent online appearances, and setting the target *prompt* data availability usually to $0.99 - 0.9999$ [3]. We decide to apply erasure coding in our system design in order to mitigate issues caused by unavailability of storing peers, with the goal of ensuring that, in a given percentage of the cases, the backup can be restored within a given timeframe with realistic settings, in which the up/download bandwidths do not allow for "instant" data recovery. As an indicative measure, we relate the prompt data availability, i.e., being able to restore all data *promptly* assuming infinite bandwidth, as an extreme to our realistic goal, which is to reach relatively low TTR values, our major measure of quality of service.

A *backup object* is the first level entity of a two-level data structure in erasure coding schemes, and its application's goal is to mitigate the problem of low number of peers while dividing a peer's whole backup data into small-sized *fragments* (the second level entity) would require many of them. Introducing backup objects allows to have small fragment size without requiring enormous number of peers, because two fragments of the same backup object cannot be stored on a given remote peer as a violation of the redundancy scheme (based on independent storage peers). The disadvantage of having large fragments is that when a remote peer goes offline before completing the upload of a fragment (which happens with higher probability if the fragment is larger), the uploading peer has to either wait for the remote peer to come back online or consider the uploaded amount to be wasted. Nevertheless, if some of a peer's fragments are uploaded to the same remote peer, the availability of the whole backup will be dependent on the availability of backup objects. That is, introducing backup objects is beneficial only if the availability of *separate* backup objects is the target metric: fragment transfers are finished earlier, and individual backup object's availability can be enhanced. Our objective, however, is to assure the availability of the whole backup data of a peer, therefore there is no point in organizing it into different backup objects. In fact, introducing backup objects is analogous to having different users (that cannot store their fragments on each other) instead of one user, each having one backup object, sharing bandwidth and storage capacity.

Data that is backed up on the data center does not need redundancy, therefore the data center will never store more fragments for a given peer, than the necessary number of fragments to restore the peer's backup and no derivative of data stored on the data center needs to be stored on peers as well. On the other hand,

the part of data stored on peers must be redundant with a redundancy rate tuned to the quality of peers. A peer needs to decide about how much part it wants to put on the data center and on peers prior to uploading the data, since the redundant coding rate cannot be increased to generate additional redundant fragments on the fly (they would not be compatible with the already existing fragments).[6] Our solution is the following: each peer makes a pessimistic estimate by constructing many redundant fragments, and uploads as many as possible to peers, and the necessary number of fragments to the data center (in PA systems) in order to always have sufficient online fragments. At the end of the process the peer deletes the remaining fragments (it may always drop fragments without violating the redundancy scheme): we adjust the number of each peer's fragments stored on the data center to the number of its fragments that have been successfully stored on the peer set, fragments on the data center are deleted if the remaining less fragments and the fragments on the peer set altogether guarantee the aimed theoretical data prompt availability.

The coding rate is chosen among the following options:

- based on system's average peer availability (when peer selection is random);
- based on the peer's own availability (when peer selection is selective).

3.3 The upload scheduling problem

The scheduling problem raises the question of how to organize data upload toward peers (in PA and PP systems). In additional, in PA systems the upload capacity must be shared between uploads to data center and to remote peers. We study multiple options:

- Upload bandwidth is focused to a single remote peer (the rationale is that the objective of a peer is to finalize the upload of a complete fragment as soon as possible) or multiple upload connections are allowed at a time (as in BitTorrent, the peers assign the set of remote peers they transfer data to/from within a given timeframe and the size of these sets is limited by the maximal parallel number of up/download connections). If multiple connections are allowed, we adapt the maximal number of parallel connections in function of the peer bandwidth: it is given by the default values multiplied by ratio of the peer's bandwidth and the average bandwidth of peers. This policy helps to avoid bandwidth underutilization at high bandwidth peers due to low limit on parallel connections.
- The scheduling strategy prioritizes upload toward remote peer(s) to which the un-transfered data parts are the least (completing fragment transfers greedily) or toward peer(s) with the highest availability.
- The scheduling strategy prioritizes upload toward remote peer(s) and transferring data to data center is performed only with excess upload capacity or when its not possible to upload toward peers (in order to limit central costs) or uploading to data center has priority (in order to achieve safe backup as soon as possible).

The pseudo-code of the scheduling algorithm with aforementioned design options is showed in Alg. 3.3. The notations are the following:

- n: number of maximal parallel uploads to peers;
- e: fragment transfer completion percentage;
- a: peer availability;
- $s \in \{e, a\}$: priority to largest transferred parts (e) or to high availability peers (a);
- d: priority to upload to data center, boolean;
- p: peer id;
- f: fragment id;

[6] The necessary redundancy rate is based on the number of peers with excess capacity, the storage capacity and the availability of peers in count. If the redundancy rate needs to be altered due to unforeseen poor peer availability, storage capacity, and/or decreased number of available peers, the coding must be re-done, since it cannot be modified during the data transfer process.

- U: set of fragments not associated with any peer;
- P: set of partially uploaded fragments, represented as a list of (p, a, f, e) quadruples, sorted by decreasing completion percentage (and peer availability if $s == a$);
- D: set of fragments uploaded to data center, represented by fragment ids f; k: number of fragments that are necessary to restore backup;
- J: set of jobs currently running represented as $(p,\ f)$ doubles, interrupted jobs (if the peer or the remote peer p goes offline) must be eliminated from J.

3.4 The data maintenance problem

In PA and PP systems data stored on peers might get lost. The problem formalizes the questions of how and when data repairs should be carried out. Our approach dictates that repair must be performed by the data owner: a peer is notified by the tracker whether one or more remote peer holding data has crashed. If a peer loses a fragment on a crashed peer, it re-generates the fragment and uploads it to the same peer. If a storing peer, having been offline during a predetermined period of time called as "grace period", is supposed to be definitely dead, the fragments it stored are uploaded to new peers or to the data center. If the "drop option" is set, peers and the data center will drop the dead peers' backup after the grace period. The grace period and the redundancy rate must be well-combined in order to be able to react fast to data loss, but without many unnecessary actions triggered by false positives.

In case of "disk crash", the crashed peer's utmost priority is to retrieve sufficient number of fragments efficiently. The peers storing fragments on the crashed peer will upload their newly re-generated fragments when the crashed peer initiates the action; TTR values shown later prove that storing peers do not need to wait long before re-uploading their data.

The default objective is to maximize the probability of being able to retrieve backup data. In order to achieve this, our implemented policy is to use as much bandwidth as the crashed peer can from its remote peers, and then saturate its downlink with downloads from the data center. This is motivated by the fact that peers' upload is the bottleneck, and it is at the peers where backup data may be easily lost, furthermore on the data center the backup is considered safe. Among the online peers, the peer choses to download its own fragments from those that have already uploaded the largest parts. Once the *necessary* number of fragments are downloaded, thus its own backup can be restored, the peer will re-download its peers' fragments that it stored and lost.

4 Simulation-based evaluation

We built a discrete Matlab simulator in order to numerically evaluate the system choices listed in Section 3. The simulation is synchronous and organized into rounds, where 1 round represents 5 minutes. Total number of rounds is 28000, thus representing more than 3 months duration. The default number of peers is 870, however in order to measure how performance varies with system scale, we perform simulations with $435, 100$ and 10 peers.

4.1 System design parameters

While the goal of our work is to perform comparing evaluation of three peer selection schemes, we emphasize the importance of the right choice of data structure and transfer scheduling schemes, that are closely related to data placement. As default setting, the number of original fragments per peer's backup data is 32, therefore with 10 GB backup data the fragment size is 320 MB. We also investigate the effects of generating large and small fragments: with a number of fragments of $8, 128, 512$, the fragment size is $1280, 80, 20$ MB respectively. Note however, that the maximal number of fragments is limited by the possible peer set size, therefore in our settings the fragment size has a lower bound which makes the discrete simulation valid.[7] We compute

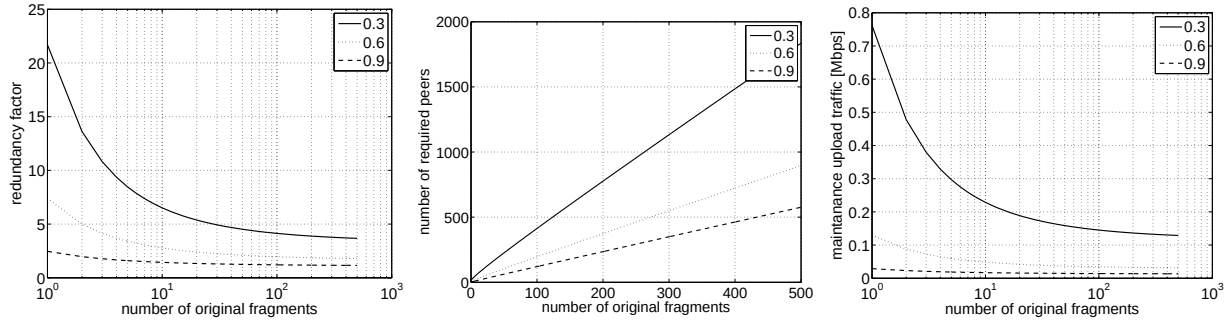
[7] BitTorrent uses 256 kB sized fragments which are orders of magnitude smaller than what is implemented in our simulation; one might consider backup data growing as a stream and performing erasure coding of reasonably-sized "data pieces" when a peer creates e.g., 1 MB backup data, then "stream" the encoded pieces to peers in order to append them to the fragments.

Algorithm 1 Scheduling data fragment upload

```
1: function updateJobs(n, s, d)
2:     while U ∪ P ≠ 0 do
3:         update J
4:         if d == true then
5:             while ‖D‖ < k do
6:                 f = pop(U ∪ P)
7:                 if f ∉ D then
8:                     upload (data center, f)
9:             d = false
10:        else
11:            ∀f ∈ P s. t. f's transfer has finished: P = P \ f
12:            ∀f ∈ P s. t. timeout on f's transfer has expired: P = P \ f, U = U ∪ f
13:            if ‖J‖ < n then
14:                upload nextJob(U, P)
15:            if ‖D‖ is not sufficient then          ▷ Not enough fragments are stored on peers yet
16:                f = pop(U ∪ P)
17:                if f ∉ D then
18:                    upload (data center, f)
19:            else
20:                if ‖D‖ is more than sufficient with fragmetns stored on peers then
21:                    f = pop(D)
22:                    D = D \ f

23: function nextJob(U, P)                            ▷ Returns: a (storer id, fragment id) pair.
24:     for all (p, a, f, e) ∈ P do
25:         if p is available for data transfer & p ∉ J then    ▷ Try to resume a partial fragment transfer at first
26:             J = J ∪ (p, f)
27:             return (p, f)
28:     if U ≠ ∅ && p = get_storer() ≠ null then          ▷ No peer with partial data online
29:         f = pop(U)
30:         U = U \ f, P = P ∪ f
31:         J = J ∪ (p, f)
32:         return (p, f)
33:     return null

34: function getStorer( )                             ▷ Returns: the identifier of a storer peer
35:     for all peer p in the system (or in the same cluster) do
36:         if p is available for data transfer and not storing local data then
37:             return p
38:     return null
```



(a) Necessary redundancy factor for 0.99 data prompt availability with various peer availabilities

(b) Necessary number of peers for 0.99 data prompt availability with various peer availabilities

(c) Maintenance traffic for 0.99 data prompt availability with various peer availabilities assuming the upload of the whole redundant data (10 GB original backup) during a disk lifetime

Figure 1: The effects of various fragment sizes

the redundancy rate in order to provide 0.99 prompt data availability if fragments are stored on peers with independent online appearances (Fig. 1). This is considered as a heuristic rule of thumb inspired by related work (data redundancy rate is determined in [3] in function of average peer availability and targeting a system-wide data prompt availability).

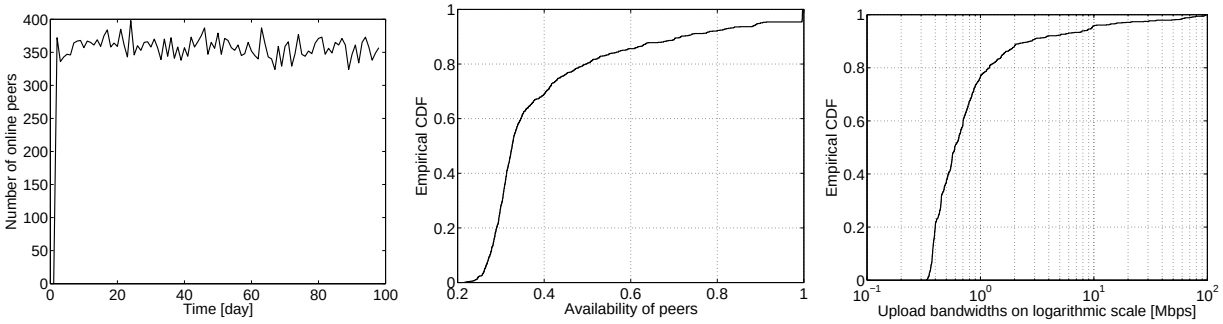
At the beginning of each round the peers assign the set of remote peers they will transfer data to/from within the round. After establishing data transfer "channels", the actual amount of data transferred between peers is determined by the bandwidth characteristics of the two parties, and the bandwidth division among the parallel connections (the number of uploads and downloads of each peer): the minimum of the respective values of two peers having a transfer channel yields the throughput. Transferring data to/from the data center is performed as a possible additional upload/download thread with saturating one's uplink/downlink.

Each peer may initiate transfers in each round to any actually online peer (up to the parallel connection limit), but does not consider possible additional uploads if one upload is finished within a round. To ensure that the discrete round length and the lack of bandwidth re-allocation within a single round after the end of a fragment transfer does not introduce distortions, we closely observe the time *and* system average of unused bandwidth fraction of peers in rounds when their maximal number of parallel up/download connections is attained, and at least one fragment transfer is finished, and they still have unfinished fragment transfers not being active in the given round, although the affected remote peer is online.

4.2 Peer parameters

The generation of synthetic traces in order to model peer dynamics is done as follows. We model peer behavior with a Markov-chain approach with peers having three states: online, offline and crashed. Three parameters, i.e., the rates of connection, disconnection and disk failure, determine the behavior of the chain. Peers' online and offline session lengths are represented in rounds, and the number of rounds within an online/offline session follows geometric distribution ("discrete exponential distribution"), generated in the following manner. For each peer random values are drawn for consecutive online and offline sessions from an exponential distribution, parametrized by the average availability of a peer's real-life trace sample from an instant messaging application, and then rounded to discrete values of rounds (Fig. 2). The average peer availabilities under 0.25 are filtered out from the traces, and all the remaining values are rounded to multiples of 0.1. Since we assume one connection/disconnection cycle per day, the average offline session length parameter equals to (24 hrs - the average online session length). Peers are launched being offline. Peers are assumed to never leave the system (during the simulation), and disk lifetimes are randomly drawn from exponential distributions with $\mathbb{E}(\text{lifetime}) = 90$ days, but we assume no crash until at least the number of original fragments have been uploaded.

The up/down link capacities are also trace-based: the uplink capacity values are randomly drawn from a real-life sample distribution (Fig. 2), and we assume for downlink capacities to be 4 times larger than uplinks



(a) Number of online peers (b) Peer availabilities (c) Upload bandwidth of peers

Figure 2: Number of online peers, distribution of peer availabilities and uplink capacities

(capped by 100 Mbps).

In order to assign a demand to each simulated peer (i.e., how much data it wants to backup), we set up storage requirement models: distribution of the storage requirements is homogeneous (every user has 10 GB backup data) or truncated power-law (with skewness parameters $0.5, 1, 2$). Similarly, local storage capacity (i.e., how much data a peer can hold) models are given by multiple distributions of the storage capacities: homogeneous (every user has 50GB capacity) or truncated power-law (with skewness parameters $0.5, 1, 2$).

4.3 Data placement scheme results — Role of the data center

Here we present intuitive expectations and obtained results about the three main data placement schemes with respect to our metrics in the system. Note, that while the following results are reached by default settings (i.e., random peer selection, data redundancy based on system average peer availability, 32 original fragments, greedy fragment completing uploads to peers, only one upload at a time, no timeout on transfers), we explore the parameter space later in more depth. Furthermore we evaluate the two data placement techniques in PA systems: one that gives priority to uploads toward the data center (uploading toward remote peers is started only after having completed the transfer of sufficient fragments to the data center in order to have a full backup) in order to yield the shortest TTB possible; and the other extrema that performs upload to data center only with excess bandwidth (when remote peers are offline and/or online remote peers have poor connectivity).

- DC: no data management and traffic scheduling are needed; all original fragments are stored on the server; fast backup, fast recovery: only the data owner's availability and up/download bandwidth set constraints.
- PA: data placement and traffic scheduling are important in order to find well-suited balance between fast backup and central costs; data transfers toward peers may take long due to poor peer availability and connectivity; peers that are rarely found online receive less storage load, however the load on peers becomes balanced in case of saturating total capacity; storing on data center mitigates the issue of long backup operations; data load on the data center grows rapidly due to the slow transfers toward peers, then decreases as more fragments are stored on remote peers.
- PP: feasible only if the peers' resource contributions can support the safe storage of the total backup demand; constructing small fragments mitigates the problem of limited storage capacity (because lower redundancy rate results in the same prompt data availability); constructing large fragments mitigates the problem of limited sized system set.

We make the differences more visible by comparing the main metrics of the three different schemes in Fig. 3 that shows plots of TTB, central costs and TTR in the systems.

- TTB: the number of rounds that elapse from the start of the peer's first fragment's upload until the completion of uploading sufficient number of fragments (which is k, the number of original fragments

of the backup data, in DC; m fragments on the data center and n fragments on peers, altogether yielding the targeted theoretical prompt data availability with their respective availabilities in PA; rk fragments, r being the calculated redundancy rate, in PP. Intuitively the TTB relation in different systems is $DC \leq PA \leq PP$ since uploading redundant backup data is limited by the overlapping online times of peers in PP, while in PA systems uploading to data center is only constrained by the peer's availability and uplink capacity (pessimistic PA's TTB is equal to DC's). For visualizing purposes the TTB is set to the simulation length for those peers that never reach the sufficient number of completely uploaded fragments due to frequent crashes of remote peers (in PP). We show the distribution of durations until the number of original fragments are uploaded completely. After this phase peers build redundancy in PA and PP cases. Then, after reaching the TTB (when the target data redundancy is obtained), further fragment transfers toward remote peers are carried out *only* with cost-minimizing purpose: storing more fragments on peers for free, and deleting fragments from the data center in exchange cuts a peer's cost that is paid for the backup service at the data center.

- Costs of central storage are measured in GBs of data placed on the data center, and as the fraction of the overall backup load. Recall, that long-term storage costs are larger than bandwidth costs, therefore traffic of data center, i.e., upload/download network traffic measured in Mbps, is directly presented by the central storage. The intuitive relation of storage costs is $DC \geq PA \geq PP$ (PA costs depend on data placement "scheduling" decisions[8]; the shortened TTB in pessimistic PA comes with the price of a transitional increased load on the data center.

- TTR, intuitively $DC \leq PA \leq PP$, is increasing by putting more and more load on peers. In DC, one can pinpoint a small number of peers that have extremely low TTR: these retrieving peers have very large downlink capacities, and since they remain online specifically for the retrieval duration, this latter is very short. The limited upload bandwidth, and the short online periods of the storing peers hinder fast data retrieval. Note, that TTR values are highly similar in the PA and PP, because in PA backup data amount increases on storing peers with time and approaches to PP. The relatively short TTR assures that definitive data loss happens only with negligible probability after at least the number of original fragments have been uploaded. We plot however the number of definitive losses that would happen if we let peers crash before having uploaded the number of original fragments.

4.4 Specific insights to different system designs

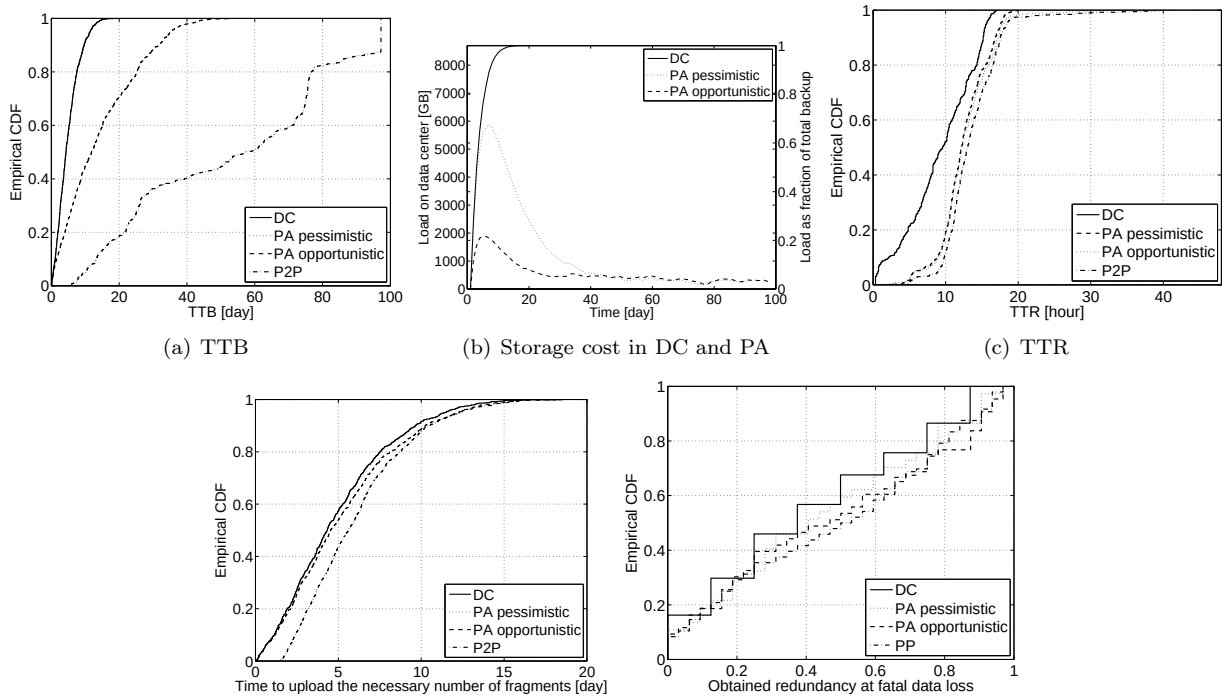
4.4.1 Peer selection

In order to support our claim on the reason of differences among TTB and TTR results, we show the negative correlation between the TTB (and TTR) and peer availability in Fig 4. Boxplots are presented in DC, opportunistic PA and PP systems for every availability group of peers; on each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. TTB values are shorter for higher available peers: the trend is more visible in PA than in DC, even more apparent in PP, especially without crashes. The reason for this phenomenon is that highly available peers are online more often, therefore capable of uploading their own backup faster. Another direct consequence of high availability is rather burdensome: since these peers are found online more often, the storage load uploaded to them is also higher than on peers with lower availability.

Motivated by the unfairness of storage load (and traffic) in a system with heterogeneous peer availabilities, we analyze in Fig. 5 the impact of clustered vs. random data placement on peers in PA and PP. In clustered data placement, selecting peers is performed based on their availability (selected peers belong to the same availability category as the data owner) and applying a redundancy rate [3] based on this availability in order to reach 0.99 prompt data availability supposing their online appearances to be independent. The applied data redundancy rates are presented in Table 1; peers that belong to the cluster of availability 1 apply the same redundancy as the cluster members of 0.9, considering the transitional outages due to peer crashes.

In general, selective data placement systems' low availability peers apply high data redundancy rate, resulting in huge amount of redundant data. The large amount of data, combined with the short overlapping

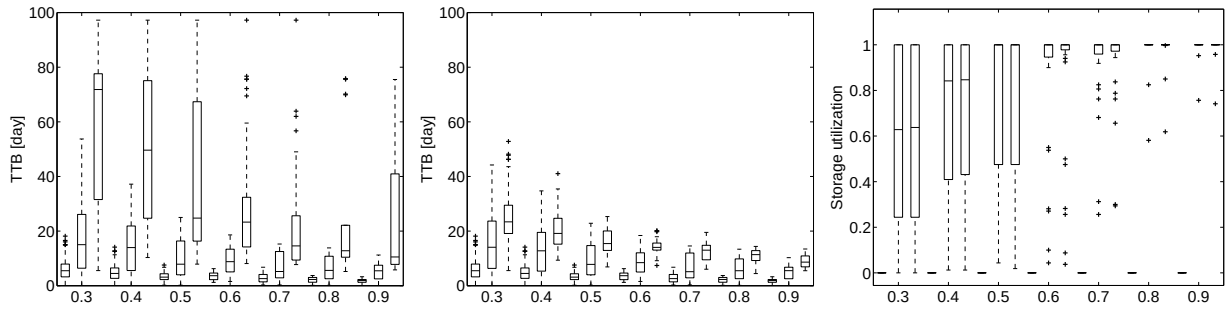
[8]The fraction of data stored on servers may be limited by a budget-threshold.



(a) TTB (b) Storage cost in DC and PA (c) TTR

(d) Time to reach redundancy 1 (e) Data loss vs. attained redundancy

Figure 3: TTB, storage cost and TTR in the data placement schemes with default settings

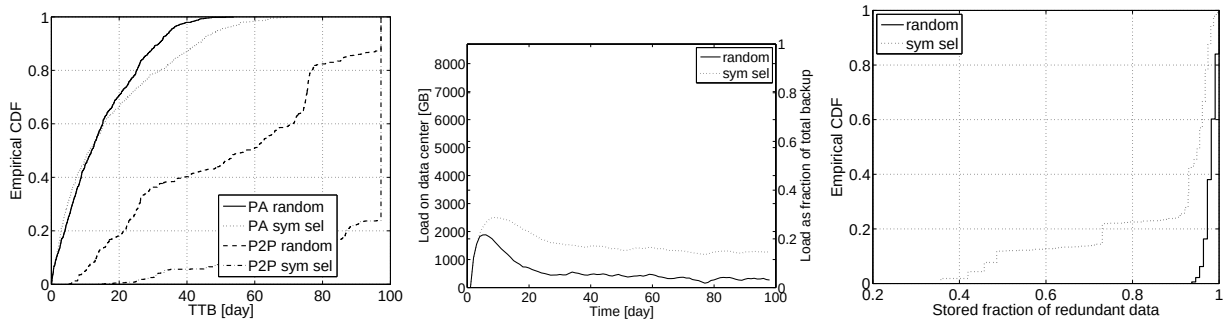


(a) TTB (b) TTB without crashes (c) Storage load on peers

Figure 4: TTB and storage burden on peers at the end of the simulation, grouped by the availability of peers, in DC, PA opportunistic and PP systems from left to right

availability	0.3	0.4	0.5	0.6	0.7	0.8	0.9
redundancy rate	4.8709	3.5529	2.7573	2.2217	1.8328	1.5323	1.2833
number of peers	554	108	66	36	28	22	56

Table 1: Redundancy rates applied with different peer availabilities and the number of peers in the clusters



(a) TTB (b) Storage cost in PA (c) Uploaded data in PP

Figure 5: TTB and storage cost in random and selective PA, PP systems

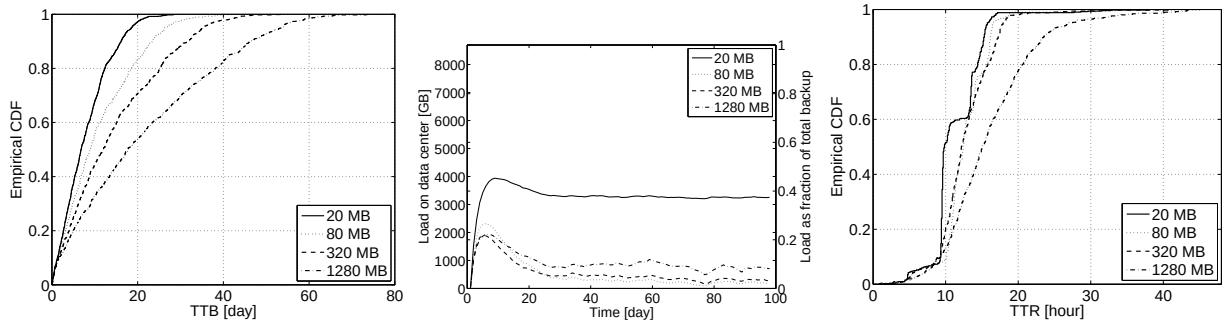
online periods (since both data owner and storer are low availability peers), makes TTB very long. Therefore, in backup systems applying selective data placement policy, the presence of the data center is crucial for peers having low availability in order to make the backup service viable. In PA the TTB is lower with random scheme except for a small fraction of peers, the high available ones. In PP, due to frequent crashes and the limited number of peers in the clusters, many peers cannot complete the backup operation.

The storage load of the data center greatly increases at the beginning in opportunistic PA, then it significantly decreases as more fragments are stored on remote peers. The un-outsourced storage amount depends on the clusters' size (beside the peers' storage capacity): in the random peer selection case the system's size does not cause such strict constraints as the cluster sizes in the selective scheme. We call the difference between the two schemes' data center storage the *price of fairness*. The great majority of peers succeed to upload almost all (more than 90%) of their redundant data in PP, and more peers have rather small success of storing all of their fragments in the selective case. This is again due to the limited number of peers within their clusters.

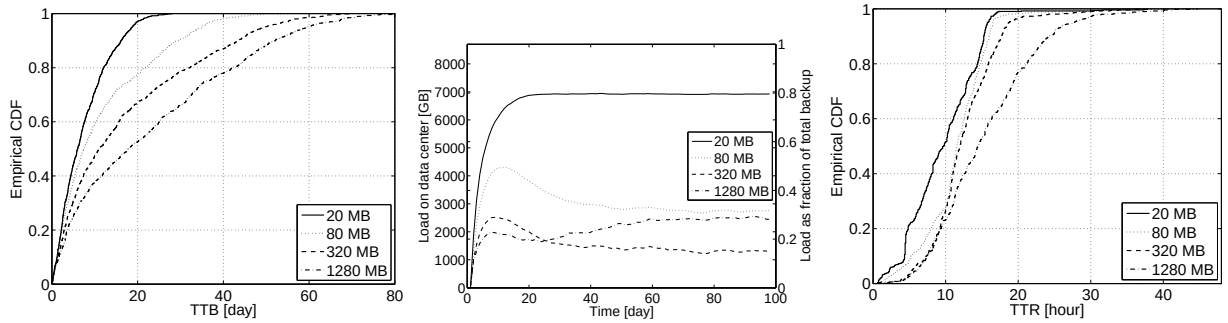
Regarding TTR, highly available peers obviously benefit from the selective scheme by being able to retrieve from their peers most of the time although constrained by the upload capacity of the few storing remote peers, on the other hand low availability peers fight short overlap durations with placing many redundant fragments on many peers, therefore widening their choice of download sources. In this sense, storing on selected peers with heterogeneous redundancy rates, the scheme adapts well to heterogeneous peers and yields good TTR results despite poor peer availability. All data placement schemes show similar performance both in PA and in PP to what have been presented earlier: the TTR does not seem to be affected by the impossibility of storing the whole redundant data on peers at this applied redundancy rate.

4.4.2 Fragment size

The effects of constructing smaller fragments in opportunistic PA are plotted in Fig. 6. Generally having small fragments the aggregate redundant data amount is less if erasure coding is applied to reach the same prompt data redundancy (see in Fig. 1), but more redundant fragments require larger peer set, which implies that in limited setting some data cannot be stored on peers (instead stored on the data center in PA). In PA, the smaller the fragments are, the shorter TTR and TTB values are yielded. Observing the centrally stored data amount, this is partly due to the fact that the peer set is limited, therefore larger data load is backed up in the data center when small, but many fragments are generated. However, as seen in the random peer selection case where the number of eligible peers does not raise strict constraints, in terms of central storage costs it is beneficial to set the fragment size to the minimal possible value that is allowed by the potential peer set size, since the amount of centrally stored data can be higher if fewer, but larger fragments are stored on the data center. Choosing a well-adapted fragment size balances the trade-off between TTB and central storage costs in function of the number of peers: uploading small fragments requires waiting to resume interrupted transfers less frequently, therefore the sufficient number of fragments can be uploaded sooner, if the peer set size allows it.



(a) TTB in PA random (b) Storage cost in PA random (c) TTR in PA random



(d) TTB in PA symmetric selective (e) Storage cost in PA symmetric selective (f) TTR in PA symmetric selective

Figure 6: Various fragment sizes in random and selective PA systems

4.4.3 Transfer scheduling

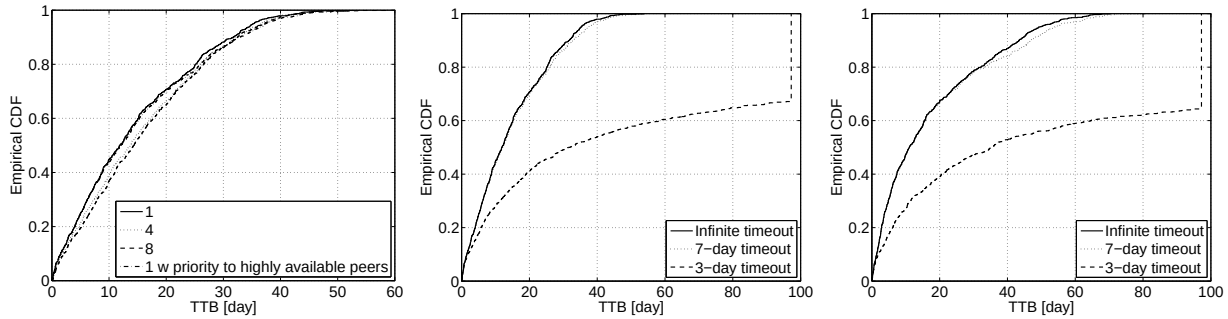
In PA and PP systems the upload scheduling plays important role because the future online appearances of remote peers are not precisely predictable and therefore fragment uploads toward them need deliberate strategy in order to complete the transfers as soon as possible. First, we change the maximum parallel upload connection number (from 1 to 4 and 8, the latter being adaptively increased if the peer's uplink capacity significantly exceeds the system average); second, we modify the "greedy" priority among peers in favor for highly available peers; third, we set 7-day and 3-day timeouts on fragment transfers. We show the TTB results in opportunistic PA systems with random and selective peer selection.

As observable, the best performance is yielded when allowing for only 1 upload instead of 4 or 8 parallel uploads, furthermore, the "greedy" scheduling strategy seems to provide slightly shorter TTB than the peer availability-based priority. Interrupting fragment uploads after various, finite timeouts drives TTB results to higher levels in both random and selective PA systems (e.g., imposing 3-day timeout on transfer makes backuping impossible for many peers, because uploading to data center is not prioritized). The conclusion, therefore, is to focus uplink capacity as much as possible on completing fragment transfers, and never waste bandwidth with aborting transfers.

4.5 Specific insights to different peer parameter inputs

4.5.1 System scale

In Fig. 8 we show that the simulation's system size was reasonably chosen. We verify the scalability of the system by performing simulations with only the half of the peers, then with 100 and 10 peers in opportunistic PA random system case. As the system size grows, the central storage can be more and more outsourced and dispersed on peers. TTB and TTR values are already nearly the same when the number of peers is 435 and 870: the peer-to-peer part of the system is scalable and the data center's storage and bandwidth capacities are (assumed to be) limitless. The difference in TTB and TTR values seen in case of 10 peers is due to the very low number of peers, thus relatively more load on the data center. As the sub-linear growth



(a) TTB with various number of parallel connections in PA random

(b) TTB in PA random with various timeouts

(c) TTB in PA symmetric selective with various timeouts

Figure 7: Various traffic scheduling techniques

of the performance metrics show, the larger the system, the better it works. We also plot the average and standard deviation (as confidential interval) of the data center storage in time, and the TTB and TTR values for peers.

4.5.2 Effects of disk crashes

In Fig. 9 we show measured TTR results in DC, PA and PP. Note, that disk lifetimes are randomly drawn from an exponential distribution with $\mathbb{E}(\text{lifetime}) = 90\text{days}$; higher data loss rate would cause more intensive traffic on peers and on the data center, moreover higher rates than the data repair rate would lead to the loss of some data definitively. Increasing the rate of data loss to levels that would create such problems in high numbers seems to be unrealistic, given that with the current setting we assume 3 months for expected disk lifetime. Obviously, considering lower expected duration for peer membership, and allowing peer departures from the system would affect the data loss rate negatively.

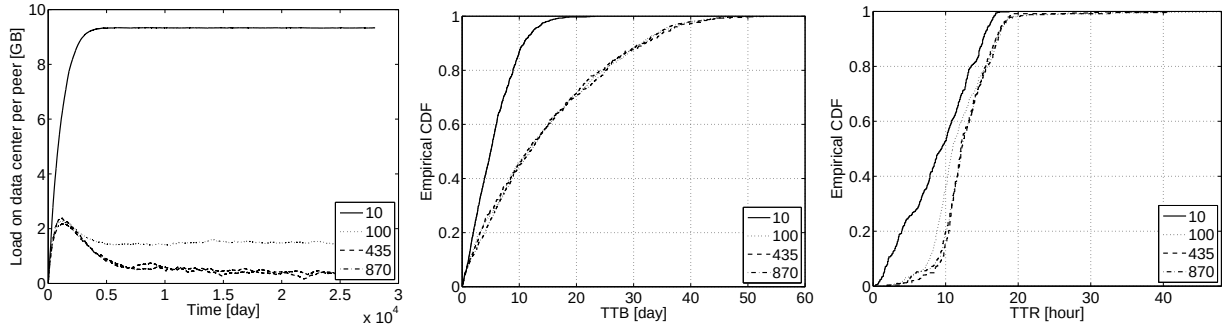
Provided that actual data loss rate takes place, we can measure the TTR values for those peers that have successfully uploaded the necessary amount of data to restore backup before they crash. Based on the plotted results, the TTR values are similar in all cases, irrespectively of the peer's availability. Central bandwidth costs due to repairs in case of disk crashes are higher if the data center's storage load is greater, i.e., in opportunistic selective PA, pessimistic PA and DC cases respectively. Since maintenance between peers "only" costs bandwidth which is considered to be free, storing less data on the data center results in savings on monetary cost of traffic as well.

4.5.3 Effects of high and low peer resources

In this section we discuss the consequences of high and low uptime, and low storage capacity contributions of peers. We show results in a "set-top-box" case, where peers attain almost 100% online availability since the backup application is assumed to be installed on their set-top-boxes that are continously connected in order to provide telephone service. On the other hand we investigate the case where only less than 0.15 average availabiities are filtered out from the default traces. For the scenario of low shared storage capacity, we suppose 10 GB of storage space instead of 50 GB at each peer.

In Fig. 10 one can observe that having outstanding or slightly less available peers than the default setting, the system's performance (especially TTB) changes significantly. Low peer availability (peers with 0.2 average availability are also present) causes higher TTB for many peers, and also increases the data center's load. Similarly to peer availability, peers' storage capacities highly influence the performance: when the setting assumes only 10 GB of shared storage capacity at each peer, the TTR is shorter, because the storage capacity provides only a slight fraction of the required capacity in order to safely store the fragments, therefore the limited peer storage capacities force peers to store on data center, hence low TTB, TTR and high data center load.

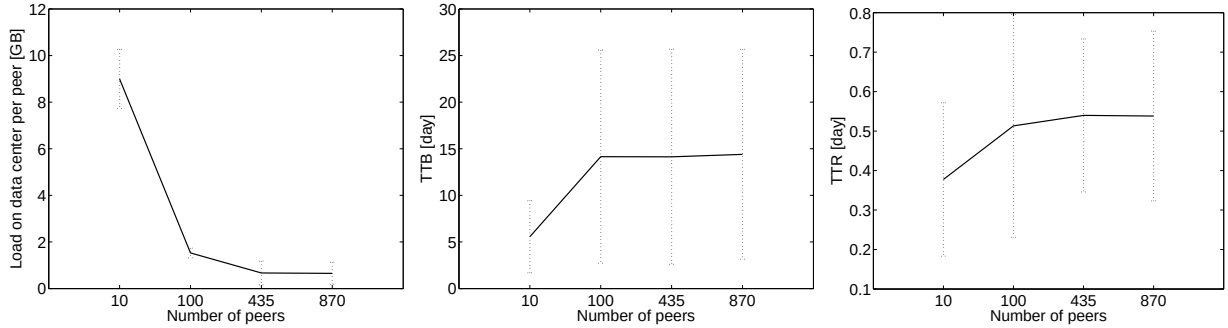
In addition to the inquiry of limited peer storage capacities, here we also consider scenarios of skewed storage demands and capacities: instead of 10 GB demand (resp. 50 GB storage), peers' demands (resp.



(a) Storage cost with various numbers of peers in PA random

(b) TTB with various numbers of peers in PA random

(c) TTR with various numbers of peers in PA random

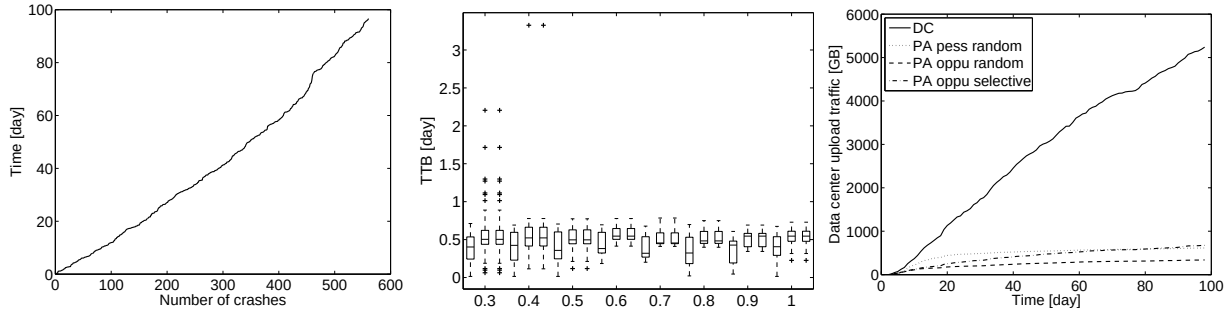


(d) Storage cost with various numbers of peers in PA random

(e) TTB with various numbers of peers in PA random

(f) TTR with various numbers of peers in PA random

Figure 8: System scalability

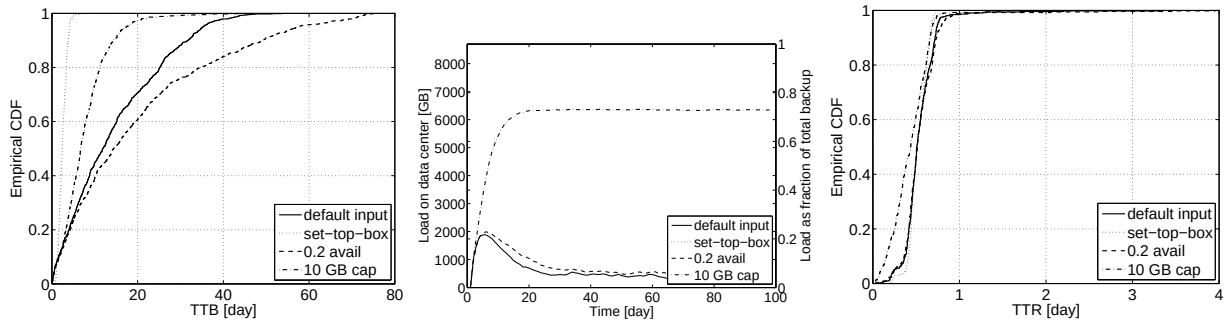


(a) Evaluation of number of crashes

(b) Measured TTR in DC, PA and PP grouped by peer availability

(c) Data center's cumulative upload traffic

Figure 9: Effects of data loss



(a) TTB (b) Storage cost (c) TTR

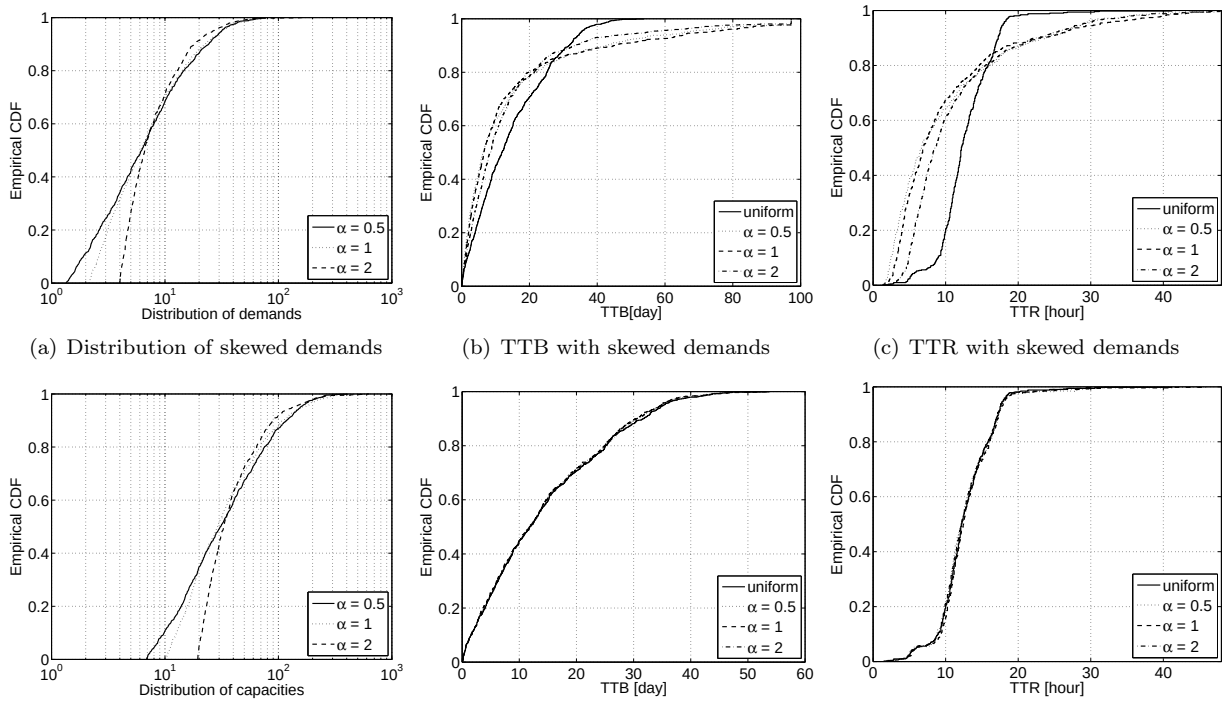
Figure 10: TTB, TTR and storage in PA random with various quality peer profiles

capacities) are randomly drawn from a truncated power-law distribution with skewness parameters $0.5, 1, 2$ and with cutoff parameter 0.1. The storage demands (resp. capacities) are normalized so that the aggregate storage demand (resp. capacity) is equal to the standard setting with 10 GB storage demand (resp. 50 GB storage capacity) at each peer. Differences in Fig. 11 are visible in TTB and TTR values with skewed demands: demand skewness worsens the TTB and TTR for a small set of peers (for those that have high demand). With differently skewed capacities, both TTB and TTR remain similar: peers that store on large capacity peers along with a lot of other peers are not constrained by the storing peer's downlink capacity, moreover crashes are not supposed to happen in bulk therefore large storage peers' uplinks do not impose constraints on retrieving from them.

5 Conclusions

We presented a broad investigation on peer-assisted online backup systems where backup data is partly placed on peers. We studied different schemes for data placement and multiple system design choices in terms of data structure and traffic scheduling. We have observed systems' performances for various user profiles, and proposed advantageous design choices. The main conclusions are the following:

- central backup service (on data center) provides the best performance in terms of backup and retrieve time but for relatively high price;
- peer assisted systems lower the storage and traffic costs, however the savings can be significant (in extremes, resulting in purely peer-to-peer system) only if peers share relatively large amounts of online availability, storage and bandwidth resources and they are sufficiently numerously present in the system, otherwise the lower costs come with intolerable loss of performance;
- applying sufficient data redundancy, the data restore duration is not significantly affected with data placement on peers, however the backup operation duration is inevitably longer in this case;
- in order to mitigate the long backup operation duration toward peers we suggest the following solutions:
  - backup to data center first (implies costs, but peak is lower than total backup, due to the outsourcing of fast uploaders);
  - apply as low redundancy rate as possible without risking definitive data loss (not necessarily based on prompt data availability);
  - construct fragments as small-sized as possible in regard to the (*required* and the *available* number of remote peers (retrieve can performed with full downlink capacity);
  - impose a low cap on the number of parallel uploads;
  - interrupt transfers only after long timeouts;
  - complete a fragment's upload greedily.



(a) Distribution of skewed demands (b) TTB with skewed demands (c) TTR with skewed demands

(d) Distribution of skewed capacities (e) TTB with skewed capacities (f) TTR with skewed capacities

Figure 11: TTB and TTR in PA random with different peer storage demand and capacity distributions

- availability-based peer selection policy yields
  - lower time to backup for highly, higher time to backup for less available peers;
  - lower average retrieval time than random placement (increased redundancy rate enhances the chance for low availability peers to find storing remote peers online during their uptime);
  - a means to avoid free-riding;

  but peers must be present in the necessary number within the clusters. Availability requirement must be extended with storage and bandwidth requirements on peers, and clustering within clusters must be prevented.

References

[1] C. G. Plaxton, R. Rajaraman, A. W. Richa: Accessing nearby copies of replicated objects in a distributed environment, *ACM SPAA*, 1997

[2] Wuala, *http://wua.la*

[3] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, G. M. Voelker: TotalRecall: System Support for Automated Availability Management, *ACM/USENIX NSDI*, 2004

[4] Amazon S3, *http://aws.amazon.com/s3/*

[5] P. Michiardi, L. Toka: Selfish neighbor selection in peer-to-peer backup and storage applications, *Euro-Par*, 2009

[6] C. Blake, R. Rodrigues: High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two, *HotOS*, 2003

[7] H. Weatherspoon, J. Kubiatowicz: Erasure Coding Vs. Replication: A Quantitative Comparison, *IPTPS*, 2002