Game-Theoretic Mechanisms to Increase Data Availability in Decentralized Storage Systems

Krzysztof Rzadca, Institute of Informatics, University of Warsaw, Poland
Anwitaman Datta, School of Computer Engineering, Nanyang Technological University, Singapore
Gunnar Kreitz, KTH Royal Institute of Technology, Sweden
Sonja Buchegger, KTH Royal Institute of Technology, Sweden

In a decentralized storage system, agents replicate each other's data in order to increase availability. Compared to organizationally-centralized solutions, such as cloud storage, a decentralized storage system requires less trust in the provider and may result in smaller monetary costs. Our system is based on reciprocal storage contracts which allow the agents to adopt to changes in their replication partners' availability (by dropping inefficient contracts and forming new contracts with other partners). The data availability provided by the system is a function of the participating agents' availability. However, a straightforward system in which agents' matching is decentralized uses the given agent availability inefficiently. As agents are autonomous, the highly-available agents form cliques replicating data between each other, which makes the system too hostile for the weakly-available newcomers. In contrast, a centralized, equitable matching is not incentive-compatible: it does not reward users for keeping their software running.

We solve this dilemma by a mixed solution: an "adoption" mechanism in which highly-available agents donate some replication space, which in turn is used to help the worst-off agents. We show that the adoption motivates agents to increase their availability (is incentive-compatible); but also that it is sufficient for acceptable data availability for weakly-available agents.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: game theory, mechanism design, distributed storage

1. INTRODUCTION

A decentralized system for data storage and replication is an important building block of many organizationally-decentralized applications, such as backup (in which the data should be durable despite of failures, e.g. symform.com, spacemonkey.com or [BitTorrentSync 2013]), or social networks [Buchegger et al. 2009] (in which, when a user is off-line, the system ensures that the user's friends can still access the data). In such systems, users store other users' data on their PCs or other devices, such as network attached storage (NAS), home routers, plug computers (such as the Raspberry Pie) or even infrastructure rented from cloud providers. Storing data consumes resources, such as disk space and bandwidth [Feldman et al. 2003]. In return, a user expects that her data will also be stored remotely, increasing availability and resilience. A generic

This paper is an extended and updated version of Krzysztof Rzadca, Anwitaman Datta, and Sonja Buchegger. 2010. Replica Placement in P2P Storage: Complexity and Game Theoretic Analyses. ICDCS 2010 Proceedings. IEEE Computer Society
This work has been supported by Foundation for Polish Science grant HOMING PLUS/2010-2/13, Swedish Foundation for Strategic Research grant SSF FFL09-0086 and the Swedish Research Council grant VR 2009-3793.

DOI:http://dx.doi.org/10.1145/0000000.0000000

storage system cannot use relations between users that are present in some applications (e.g., "friends" in a social network [Sharma et al. 2011]). Thus, users are modelled as autonomous agents [Babaioff et al. 2007; Bhagwan et al. 2003], and they seek to maximize their perceived profits (e.g., availability of their data) and to minimize their contribution (e.g., the amount of other users' data they store). The crucial decision an agent must make is to choose other agents that will replicate her data and whose data she will replicate, as we assume a reciprocity-based scheme. Such a reciprocal contract allows the agents to swiftly adopt to changes in their replicator's availability: if the other party stops serving the data, an agent will drop (or let expire) the contract and choose another replicator.

Depending on the organization of the system, the choice of replicators is either done through the agency of a centralized matching system, (like in spacemonkey.com or symform.com) or using a decentralized algorithm in which agents form replication agreements autonomously [Douceur and Wattenhofer 2001a; Bernard and Le Fessant 2009].

In this paper, we study the problem of maximization of *data availability* in a decentralized data replication system. We use a stochastic model of agent availability. We study two scenarios of matching, i.e., organizing replication agreements between agents: centralized and enforced; or decentralized and autonomous.

In order to derive worst-case bounds, we assume that agent availability is stochastic and constant in time. An agent's availability is the probability of the agent being available (correlated with the agent's expected lifetime [Bernard and Le Fessant 2009; Bhagwan et al. 2004]). The goal is to maximize data availability given the constraints on the storage size. In our earlier work [Rzadca et al. 2010], we also investigated an alternative model, in which availability was deterministic (on-line/off-line), but varied in time. The alternative model offered higher data availability when agents had varied availability patterns (corresponding to a world-spanning system); thus the model presented in this paper shows a worst-case for a more general model.

We analyze the problem when matching is done either *centrally* or in a *decentralized* manner. A *centralized* system collects information about the agents' availabilities and then derives replication groups so that the expected availability (or resource usage) is optimized in a manner equitable to all the participants. A system with central matching models existing commercial projects (symform.com, spacemonkey.com, or the early versions of wuala.com), in which agents do not control with whom they replicate their data. The matching algorithm can be run on a few nodes controlled by the system owner (for instance, a tracker). In a *decentralized* system, each agent seeks to find replication partners maximizing its own data availability. A decentralized system models users running free (open-source) software (and thus able to modify it); as the software aims to store and manipulate vast amounts of sensitive data, it is important to keep it open so that it can be validated and patched by the community.

The paper has the following contributions: (i) In centralized matching systems we prove that achieving equitable allocation is NP-complete. (ii) We propose heuristics for centralized matching to achieve equitable availability. According to our experimental evaluation, for all classes of agents this heuristic achieves data availability two to three orders of magnitude higher than random allocation.

(iii) In systems with decentralized matching, we prove the existence of a unique subgame perfect equilibrium, in which agents replicate data with other agents that have similar availability (thus, the most available agents replicate data only among themselves). Such an equilibrium may seem inefficient for the system goal, as we show an instance in which it is arbitrarily far from the equitable solution, i.e., the price of anarchy [Koutsoupias and Papadimitriou 1999] is arbitrarily high. Yet the equilibrium is fair in the sense that the highly available agents have their data replicated better

than erratic, unstable agents. (iv) We propose a distributed heuristic that according to our experimental evaluation converges fast (in the order of tens to hundreds of seconds) to the subgame perfect solution. The short convergence time means that the system adapts fast to the changes in agents' availability.

(v) We propose a set of rules for a game-theoretic mechanism that reduces the price of anarchy and, at the same time, provides incentives to agents to be highly-available. (vi) We present a heuristic that computes allocations adhering to the rules for the mechanism. According to the experimental evaluation, the heuristic increases the data availability of the worst-off agents by two to three orders of magnitude.

The paper has the following organization. Section 2 reviews the related work. Section 3 introduces the mathematical model. The centralized matching (NP-hardness, fairness, heuristics) is analyzed in Section 4. The decentralized matching (the Nash equilibrium, the price of anarchy) is analyzed in Section 5. Section 6 proposes a framework for a game-theoretic mechanism that increase the data availability of the weakly-available agents. A distributed algorithm for matching agents is presented in Section 7. Section 8 presents results of simulation of the algorithms. In simulations, we also consider a cloud-assisted system, in which a fraction of users contributes highly-available resources rented from cloud providers. Section 9 discusses relaxations of some of our assumptions. Finally, Section 10 summarizes the paper.

2. RELATED WORK

Our paper analyzes the problem of replica placement in a decentralized storage system. In order to optimize the data availability, we explicitly choose the nodes on which the data is to be replicated (unlike, e.g., DHTs [Rowstron and Druschel 2001]).

Many papers assess loss of efficiency of the system caused by selfishness of individual participants, starting from the analysis of selfish routing in [Koutsoupias and Papadimitriou 1999]. [Babaioff et al. 2007] studies incentives for system-optimal behavior in p2p systems. [Fabrikant et al. 2003] considers a game of network creation in which selfish nodes optimize the cost of creating new links that minimize distances to other nodes. [Moscibroda et al. 2006] considers a similar model in a p2p system. In contrast, in decentralized storage systems, the "transaction" (i.e., storing data of another agent) is persistent rather than temporary; unlike in the classic Prisoner's Dilemma, an agent's utility from "cheating" is small, as the cheated party can easily detect cheating and quickly adjust its strategy. Interaction is repeated — similarly to BitTorrent [Rahman et al. 2011] — thus, the typical problems of enforcing agreements can be easily solved using reciprocity and a basic reputation system; which is why in the game theoretic analysis (Section 5), we focus on the process of forming, rather than enforcing, replication agreements.

The most common approach to decentralized storage is to place *enough* replicas *randomly* in the system. [Bhagwan et al. 2002] analyzes how many replicas are required to achieve a desired level of availability. Pastiche [Cox et al. 2002] and Total Recall [Bhagwan et al. 2004] use random placement. P2P file systems using DHT for storage and routing also place replicas randomly (i.e., not taking into account nodes' properties); such systems include [Muthitacharoen et al. 2002; Busca et al. 2005; Amann et al. 2008; Zhang et al. 2007; Chang et al. 2008]. Chun et al. [Chun et al. 2006] studies by simulation durability and availability in a large scale storage system, focusing on maintenance in presence of permanent failures. Bhagwan et al. [Bhagwan et al. 2002] and Rodrigues and Liskov [Rodrigues and Liskov 2005] show basic analytical models and simulation results for data availability under replication and erasure coding.

Another common approach is to use caching to store replicas of popular files. For instance, FreeNet [Clarke et al. 2001] relies on proximity-based caching; when a data item is no longer used, it can be removed from a cache. In Pangea [Saito et al. 2002],

a replica is created whenever a data item is accessed. Other works on proximity-based caching include [Iyer et al. 2002; Shi and Mao 2006; M. Hefeeda and Mokhtarian 2008; Linga et al. 2003]. These approaches have limited applicability in a general storage system in which there probably would be a long-tail of files that are generally unpopular, yet important to some small communities.

Approaches that explicitly optimize the placement of replicas include [Douceur and Wattenhofer 2001a; 2001b; Bernard and Le Fessant 2009; Michiardi and Toka 2009; Toka and Michiardi 2011]. [Bernard and Le Fessant 2009] studies by simulation an algorithm similar to the Pragmatic Queries algorithm (Section 7), with a slightly more complex acceptance ($score(i, j)$) function. Similarly to our theoretic and simulation results, the experiments in [Bernard and Le Fessant 2009] show that highly available agents achieve better performance than the agents with lower availability.

[Michiardi and Toka 2009] and [Toka and Michiardi 2011] analyzed a problem similar to the decentralized allocation in our model. [Michiardi and Toka 2009] observes that the "system stabilizes when peers are grouped into clusters, pooling users that have similar profiles", however the proof is left for future work. In our paper, Proposition 5.2 provides a proof for the analogous behavior in our model; moreover, we compute the price of anarchy (Proposition 5.5). Additionally, we prove that the centralized optimization of availability is NP-hard (Propositions 4.2-4.5), which is hypothesized, but deferred for future work in [Toka and Michiardi 2011].

We consider estimates of availabilities that do not depend on time. An alternative is to have many estimates for future time periods (see also Section 9). In this model, [Blond et al. 2012] analyzes the problems of presence matching and uptime matching. A related model, grouping nodes with known join and leave times to cover the whole availability period, is analyzed in [Li et al. 2011]. According to our earlier work [Rzadca et al. 2010], such alternative models result in higher data availability (when agents have varied availability patters); thus the results presented in this paper may be regarded as a worst-case (a realistic worst-case when the deployment is not global).

Fair optimization of availability (Section 4) is similar to the problem considered in [Douceur and Wattenhofer 2001a; 2001b], where replicas of individual files are spread over a pool of hosts with given availabilities. These approaches were developed for Farsite [Bolosky et al. 2007], an organizationally-centralized filesystem; thus there is an implicit assumption of a single owner of the system. These methods do not consider the autonomy of the participating agents.

[Giroire et al. 2009] assumes that the replica placement policy is partly determined by locality constraints. The "Buddy" policy is similar to our clique-based allocation; however our cliques are optimized, and not partly fixed. By simulation, the authors conclude that locality-aware policies are less efficient than the global, randomized allocation; in contrast, our Equitable heuristic is more efficient than the random allocation.

[Raz et al. 2008] models a system with storage clients and storage servers and proposes an agent-based algorithm to match clients' requirements to servers' offers. Clients pay for servers' services. In our system, there are no "servers": each agent both offers storage space for others and stores its data on others' devices. We are therefore able to construct a system without monetary payments, which could have negative consequences for users [Vohs et al. 2006]. The optimization protocol proposed in [Raz et al. 2008] has some similarities to the distributed algorithm we propose in Section 7 in that a client and a server make autonomous decisions to optimize their performance. However, [Raz et al. 2008] uses random matching, while we use T-Man [Jelasity and Babaoglu 2006] as a distributed global optimization sub-protocol.

Table I. Summary of notation used in the paper

p_i, i	agent
u_i	an estimate of agent's (true) unavailability
u	a vector of agents' unavailabilities, $u[i] = u_i$
u'_i	agent's declared unavailability
$u_i(j)$	unavailability of agent j's data on agent i
d	data unavailability function
d_i	data unavailability of an agent i
s	agent's suplus units of storage (used to replicate other agents' data)
G_j	replication group
$d(G_j)$, d_{Gj}	data unavailability of a replication group G_j
σ	group size ($\sigma = s + 1$)
n	the total number of agents
$r_l(i, 1, j)$	replication proposal by agent i to agent j in round l
$r_l(i, 0, j)$	withdrawal of a replication proposal in round l
r	the number of replication slots used by the mechanism
k	the number of times a weakly-available peer is adopted
τ	threshold separating weakly and highly-available peers

Our aim is to build a generic storage; a specialized system could use real-life relations between users (a friend-to-friend, F2F, system, [Sharma et al. 2011; Tinedo et al. 2012]).

We propose (Section 6) a truthful mechanism that considers only a single parameter of the system: the agents' availability. An alternative scheme is to consider also storage space. With asymmetric agreements [Pamies-Juarez et al. 2011], weakly-available agents contribute more disk space to their highly-available replication partners to compensate for their availability.

The payoff function in the replication game (Definition 5.1), in which an agent's payoff depends on its availability (type) and availabilities of its replication partners, fulfills the definition of a hedonic game [Drèze and Greenberg 1980]. However, a (general) hedonic game has a complex structure of payoffs: the agent's preferences are defined over all coalitions she might belong to. In a more restricted model, utilities are symmetric and additively-separated [Bogomolnaia and Jackson 2002]. Our replication game has an even more restricted structure of payoffs: each agent's contribution to the coalition's payoff is the same — equal to the agent's unavailability. Using this restriction, we prove results stronger than those for the generic hedonic game: in particular, the price of anarchy (Proposition 5.4) and a truthful mechanism to reduce it (Section 6).

3. SYSTEM MODEL AND ASSUMPTIONS

This section proposes a mathematical model of a decentralized storage system. The goal of the model is to focus on the consequences of the autonomy of agents, thus we simplify other issues present in the system; in Section 9 we discuss relaxations of these assumptions. The notation is summarized in Table I.

The system is composed of n autonomous agents (peers, players). The i-th agent is denoted by p_i, or alternatively by i if it is not ambiguous.

In the majority of the paper, to make the mathematical analysis tractable, we assume that agents are homogeneous in terms of storage needs and available storage resources to share. Thus, all the agents provide a surplus storage space of s units to the system and need to store one unit of data. Note that in addition to storing s units of other agents' data, each agent stores a copy of its data; thus the total storage size of each agent is $s + 1$. Our results can be generalized to systems with non-homogeneous agents: we extend the analytical results in Section 9.1; and study the impact of various storage sizes experimentally in Section 8.5.

To make mathematical analysis tractable, we assume that agents replicate data by storing complete copies, in contrast to erasure codes. Erasure codes might offer additional gain in data availability.

Agent i's *availability* a_i is the probability of being online. Agent i's *unavailability* representing the probability that agent i is not available is given by $u_i \triangleq 1 - a_i$. Later on, we use unavailabilities rather than availabilities as it simplifies all the mathematical expressions.

The availability estimates can be measured by a trusted centralized component, or by agents themselves [Le Fessant et al. 2008]. Thus, we assume that the availabilities are publicly known and cannot be tampered with by agents.

Our key contribution is a series of algorithms organizing agents into replication groups. These algorithms, given the estimates of agents' unavailabilities (u_i), produce replication groups to optimize various performance measures. For the sake of tractability of the analysis, we study a snapshot of the system and thus we assume that these estimates do not change: an algorithm, given the best guess on future agents' performance (u_i), organizes replication. Adopting the algorithms (or the solution) to changing availabilities is out of the scope of this paper (although we empirically study performance under changing availabilities in Section 8.6 and give some further ideas in Section 9). Additionally, our matching algorithms converge fast (in order of tens of seconds to minutes); this is shorter than the detection of any significant availability change.

We also assume that the estimate of the availability is not a function of time; an alternative model is to consider that estimate depends on the (wall-clock) time, modeling e.g. daily usage patterns of a device [Rzadca et al. 2010].

In order to simplify the notation of proofs, we assume that agents are numbered according to increasing unavailabilities, $u_i \leq u_{i+1}$. We also assume that n is divisible by $s+1$ (otherwise, at most s "virtual" agents with $a_i = 0$ can be added to the system—these agents represent the replication slots that will be "wasted" in a system).

We model a storage system based on Internet connections, thus we assume that the agents are connected by a complete graph: any agent can contact and replicate data at any other agent. We also assume that the bandwidth is not a bottleneck — see Section 9 for a discussion on how the bandwidth might be modelled.

Assume that i's data is replicated by agents $\{p_{i1}, p_{i2}, \ldots, p_{ik}\}$, where $p_{i1} = i$. i's *data* is thus unavailable with probability d_i equal to the probability of the event that all the replicators are offline (a transient failure), $d_i = \prod_{j=i1\ldots ik} u_j$ (assuming that the agents' transient failures are independent [Babaioff et al. 2007; Bhagwan et al. 2003]).

In several of our models, we assume that agents form replication *groups* (*cliques*) $\{G_j\}$. Each member $p_i \in G_j$ of the group replicates data of all other members $p_j \neq p_i \colon p_j \in G_j$. We define group unavailability $d(G_j) \triangleq \prod_{i \in G_j} u_i$. Note that, $\forall p_i \in G_j \colon d_i = d(G_j)$. We denote group size as $\sigma_j = |G_j|$. We do not use groups in the availability-encouraging mechanisms in Section 6. Such groups have several advantages compared with the pair-based allocation. First, groups are formed in the subgame perfect solution of the decentralized versions of the problem (see Proposition 5.2 that considers the problem without assuming replication groups and proves that such groups will be formed). Second, with groups, it is easier to optimize some of the system's parameters not directly considered in this paper, like data dissemination during updates, when a group can form a spanning tree. Third, as groups are based on reciprocity, it is easier for agents to directly control their replicas and react to free-riding.

As agents are assumed to be rational and to derive utility from availability of their data, each agent wants its data to be replicated as well as possible. Thus, i minimizes its d_i by choosing, or forming, a group with $d(G_j)$ as small as possible. Depending on the constraints of the system, this goal can be expressed in two ways:

(1) given the surplus storage of s units, find s agents $\{p_{i2}, \ldots, p_{i(s+1)}\}$ such that $d(G_j) = d(\{i\} \cup \{p_{i2}, \ldots, p_{i(s+1)}\})$ is minimized (this goal represents the optimization of data availability given a budget on the storage space);
(2) given the maximal tolerable unavailability ϵ, minimize the size of the replicating group $\min \sigma_j = |G_j|$ (this goal represents the optimization of the storage space, given a budget on the availability).

Both types of goals can be expressed also from the system's perspective. The system should guarantee that agents are treated fairly by optimizing unavailabilities of *all* the groups $\{u(G_j)\}$. In this paper, instead of a multi-objective approach, we will aggregate the groups' goals using two aggregating functions: (i) $\sum d(G_j)$ (expressing the average-case behavior); (ii) $\max d(G_j)$ (expressing the worst-off group). Section 4.2 provides motivation for aggregating over groups, rather than individual agents.

4. CENTRALIZED MODEL: COMPLEXITY, BOUNDS AND HEURISTICS

This section analyzes the model with stochastic agent availabilities from the perspective of a centralized system that organizes the replication agreements. First, we prove that equitable optimization of the availability is NP-hard. Then, we propose aggregation functions used to characterize fairness of the system; and propose lower bounds which we later use for the price of anarchy estimation. Finally, we show a simple, greedy heuristic that organizes agents into cliques.

4.1. Complexity of Centralized Matching

The task of the fair matching system is to divide agents into replicating groups such that: (i) the probability of data being online is as high as possible; (ii) the size of each group is bounded. In this section, we first define a simple version of this replication problem and prove that it is NP-complete. Then, using this result, we prove that both system-level problems are NP-hard (namely, optimizing availability given constraints on the storage, and optimizing group size given minimal availability).

We define a simplified version of the replication problem as follows.

Definition 4.1. An instance of the decision version of a Simple Stochastic Fair Replication Problem (SSFRP) is given by the set of the agents' unavailabilities $\{u_i\}$ and a bound B. We ask whether there is a partition G_1, G_2 of agents' population such that both groups are non-empty and $d(G_1) + d(G_2) \leq B$.

This problem is "simple", because we consider only two groups, we do not consider limits on the size of each group and we use a simple sum as an aggregation of groups' unavailabilities.

PROPOSITION 4.2. *The decision version of the Simple Stochastic Fair Replication Problem is NP-complete.*

PROOF. (Sketch, full proof in the appendix) Reduction from Partition with $u_i = 2^{-b_i}$, $B = 2 \cdot 2^{-S/2}$. □

As the most unrestricted version of the replication problem is NP-complete, other problems are similarly NP-complete. Maximizing availability with constrained resources (Optimum Availability with Constrained Storage, OACS) translates into a problem of forming groups with a limited number of members. SSFRP can be thus solved by OACS with unlimited groups.

COROLLARY 4.3. *The problem of optimizing availability given the maximum number of agents in each group (OACS) is NP-complete.*

Restricted size problems are, generally, harder than non-restricted ones. For instance, creating groups with exactly 3 members corresponds to the strongly NP-complete 3-partition problem [Ausiello et al. 1999].

Finally, we consider the problem of minimizing the used storage space, given a constraint on the minimal availability of a group (Optimum Storage with Constrained Availability, OSCA). OSCA is solved by forming the maximum number of groups such that each group provides at least the required availability level R.

Definition 4.4. The Decision version of OSCA is defined as follows. Given the agents' unavailabilities $\{u_i\}$, we ask whether it is possible to construct a partition of agents' population consisting of at least N disjoint groups $\{G_1, \ldots, G_N\}$, so that in each group G_j $\prod_{i \in G_j} u_i \leq R$.

PROPOSITION 4.5. *OSCA is NP-complete.*

PROOF. (Sketch, full proof in the Appendix) Reduction from DUAL BIN PACKING [Assmann et al. 1984] with $u_i = 2^{-b_i}$ and $R = 2^{-B}$. □

4.2. Characterization of a Fair Allocation

In multi-agent optimization, each agent has an associated outcome function f_i. In order to optimize performance of all agents, but to avoid multi-objective optimization methods, these functions are aggregated (commonly as $\sum f_i$ or $\max f_i$) which transforms the problem to a single-goal optimization ($\min \sum f_i$, $\min \max f_i$). The solution obviously depends on the aggregation function: for instance, $\max f_i$ focuses on the worst-off agent, at the expense of others; $\sum f_i$ ignores what was the base on which an improvement in one of f_i was made.

In the subsequent analysis, in order to make our results more general, we will use two "fair" aggregations: $\sum_i d_i$ (the average replication level of an agent) and $\max d_i$ (the worst replication level). To further simplify the formulation of some proofs, we will aggregate over groups, rather than over individual agents. $\max d_i = \max_G d(G)$. Similarly, $\sum_{G_j} d(G_j)$ can be used instead of $\sum_i d_i$: a corollary from Proposition 4.6 is that $\sum_i d_i = \sigma \sum_{G_j} d(G_j)$.

As the problem of deriving a fair allocation is NP-hard, in this section we show some properties of the optimal fair allocation. These properties will help to derive the price of anarchy in Section 5.

The following two propositions state that it is sufficient to analyze allocations in which all the replication cliques are complete (have the maximum number of agents).

PROPOSITION 4.6. *Any grouping with replication groups having fewer than $\sigma = s + 1$ agents can be transformed to a grouping with all groups having exactly σ agents and a smaller or equal value of $\sum_i d(i)$.*

PROOF. We analyze groups having less than σ members. The total number of agents across all such groups is divisible by σ. Repeat the following until all groups have exactly σ members. Take the group G with maximal $d(G)$ among these groups and assign its members to other non-full groups (non-full groups have space to accept these agents, otherwise the number of agents in these groups would not be divisible by σ). Observe that, overall, $\sum_i d(i)$ has not been increased, as former members of G are assigned to groups G' having $d(G') \leq d(G)$; and after the re-assignment, $d(G')$ is also not increased (and decreases if the reassigned agent has $u_i < 1$). □

PROPOSITION 4.7. *Any fair solution optimizing $\min \max G_j$ can be transformed to a solution with all groups having exactly σ agents (without increasing the value of the goal function).*

PROOF. The proof is similar to the proof of Proposition 4.6. □

The following proposition shows a lower bound for the $\min\max$ aggregation.

PROPOSITION 4.8. *A lower bound for* $\min\max d(G_j)$ *is* u_G^σ*, where* u_G *is the geometric average of* $\{u_i\}$*,* $u_G = \sqrt[n]{\prod u_i}$.

PROOF. The proof is by contradiction. Assume $\min\max d(G_j) < u_G^\sigma$. In such a solution, $\forall_{G_j} d(G_j) < u_G^\sigma$, thus

$$\sum_{k=1,\ldots,N} \log d(G_j) < N \log u_G^\sigma.$$

As $N = \frac{n}{\sigma}$ and $\log u_G = \frac{1}{n}\sum \log u_i$, we get

$$\sum_{k=1,\ldots,N} \log d(G_j) < \sum_{i=1,\ldots,n} \log u_i. \tag{1}$$

However, as each agent belongs to exactly one group, in any solution:

$$\sum_{k=1,\ldots,N} \log d(G_j) = \sum_{k=1,\ldots,N} \sum_{i\in G_j} \log u_i = \sum_{i=1,\ldots,n} \log u_i,$$

which contradicts (1). □

The following proposition shows a lower bound for the other considered aggregation, $\min\sum$.

PROPOSITION 4.9. *A lower bound for* $\min\sum d(G_j)$ *is* $\frac{n}{\sigma}\Big(\min_{i\in\{1,\ldots,n\}} u_i\Big)^\sigma$

PROOF. In each group G_j:

$$d(G_j) = \prod_{i\in G_j} u_i \geq \Big(\min_{i\in G_j} u_i\Big)^\sigma \geq \Big(\min_{i\in\{1,\ldots,n\}} u_i\Big)^\sigma.$$

Thus,

$$\sum d(G_j) \geq N\Big(\min_{i\in\{1,\ldots,n\}} u_i\Big)^\sigma,$$

and the proposition follows. □

4.3. A Greedy Heuristic for the Fair Allocation Problem

The following greedy heuristic optimizes the assignment of agents to cliques in OAFS (Section 4.1), assuming global knowledge and coordination of agents. The idea of the algorithm is similar to the First Fit Decreasing approximation algorithm for minimum bin packing [Ausiello et al. 1999].

Firstly, the agents are sorted by increasing unavailabilities $u(i)$. Then, $\frac{n}{\sigma}$ most available agents are assigned to separate cliques. Finally, for each of the remaining agents (in the sorted order), the agent is assigned to a non-complete clique G_j that currently has the highest unavailability $G_j = \arg\max_{Gj':|Gj'|<\sigma} \prod_{i\in Gj'} u_i$.

We experimentally compare this algorithm to the random allocation in Section 8.2. The assignment resulting from the above heuristic may be further optimized by a global search meta-heuristic, such as Simulated Annealing (SA). However, in our initial experiments, SA did not significantly improve results returned by the heuristic, probably because of the large number of cliques to consider.

5. GAME THEORETIC ANALYSIS OF THE DECENTRALIZED MATCHING

In decentralized matching, we assume that each agent is selfishly interested in maximizing the availability of its data. In this section, we predict replication agreements that will be formed in such systems. We model the resulting game as an extensive game in which agents change their replication agreements. We show that in the unique subgame-perfect equilibrium agents will form replication agreements with agents of similar availability. The drop in the system's efficiency (both for $\sum_{Gj \in \mathcal{G}} d(G_j)$ and $\max_{Gj \in \mathcal{G}} d(G_j)$) in this equilibrium is unbounded.

5.1. Replication System as a Game

As data replication is a long-lasting agreement, two distinct phases can be logically distinguished. In the first, *organizational* phase, each agent forms zero, one or more agreements with other agents in which she commits to storing their data. The data is transmitted only when this phase ends. In the second, *production* phase, the agent can either honor the previous agreements, or break some of them by explicitly dropping other agents' data or by lowering its availability (see Section 9 for a discussion on how these phases map to a dynamic system).

The game in the *production* phase of the system is similar to the repeated Prisoner's Dilemma [Osborne 2004]: an agent prefers not to honor the previous commitments, as storing data consumes agent's resources. However, the goal of the replication system is to make data available over a longer time period, thus the game can be modeled by the infinitely repeated Prisoner's Dilemma with a discount factor δ close to 1. Thus, breaking the agreements is only profitable in a very short term: when j detects that i stopped replicating j's data, j will not only break all its agreements with i, but also notify other agents of a "cheater" (directly, or with the help of a reputation system), which, in turn, can effectively exclude i from the replication system.

5.2. Definition of the Game

We formally define the game in the *organizational* phase as an extensive (multi-round) game with simultaneous moves [Osborne 2004]. Intuitively, in each round of the game, zero, one or more agents propose to replicate other agents' data and/or withdraw previous proposals. The game ends when no agent changes its set of replicating agents.

Definition 5.1. The *Stochastic Replication Game* (SRG) is defined as an extensive game with infinite horizon and simultaneous moves, in which:

— the set of *players* is equal to the set of agents;

— the set of *terminal histories* contains list of sets $(\{r_l(i, 0 \vee 1, j)\})$, i.e., sets of replication proposals (denoted by $r_l(i, 1, j)$) or withdraws of previous proposals ($r_l(i, 0, j)$, possible only when $\exists l' : r_{l'}(i, 1, j)$), made by agents ($i$) to other agents ($j$) in round l; in each round, for each agent, the number of active replication proposals does not exceed the agent's storage capacity s; all terminal histories end with an empty set $\emptyset$;

— the *player function* $\mathcal{P}(h) = \{p_i\}$, i.e., after each history h all players can make proposals;

— each player minimizes the expected unavailability of its data computed as a product of unavailabilities of players who propose replicating the player's data (and who do not withdraw their proposals in subsequent rounds). We denote by R_j the replication set of j (after a terminal history), i.e., $R_j = \{i : (\exists l_1 : r_{l1}(j, 1, i)) \wedge (\nexists l_2 > l_1 : r_{l2}(j, 0, i))\}$. The *pay-off* is $d(i) = u_i \prod_{j : i \in R_j} u_j$.

The game is defined as an extensive game to model the fact that during the organizational phase agents will react to other agents' decisions and adapt their replication

sets accordingly. Similarly, the game is not repeated, as the game models the organizational phase in which the payoff is computed for the production phase rather than for the short-term state after each round. For this theoretical analysis we do not limit the number of rounds in the game.

5.3. The Nash Equilibrium and the Price of Anarchy

We study the outcome of the game assuming that agents' strategies are tit-for-tat based, i.e., if agent i proposes to replicate j's data in round l ($r_l(i,1,j)$) and agent j does not propose to replicate i's data in the subsequent round at the latest ($\nexists l' \leq l+1\colon r_{l'}(j,1,i)$), agent i will withdraw its proposal in the next round $r_{l+2}(i,0,j)$. This assumption on strategies helps agents to coordinate their actions. At the same time, such strategies are flexible and allow agents to react to actions of other agents.

The following proposition shows the *subgame perfect* [Osborne 2004] equilibrium of the game. Every subgame perfect equilibrium is a Nash equilibrium. In extensive games, the notion of the Nash equilibrium is considered artificial, as it is based on so-called empty threats [Osborne 2004, p. 164]. In contrast, the subgame perfect equilibrium requires that each player's strategy must be optimal for every history after which the player moves. In order to illustrate the difference, assume that $s = 1$ (each agent can replicate data of only one other agent), and $u_1 < u_2 < u_3 < u_4$. If p_2 and p_3 commit to mutual replication, and p_4 has a tit-for-tat strategy and replicates p_1, in the Nash equilibrium p_1 must replicate p_4 data—after p_1 proposes to replicate p_2, p_2 would not withdraw p_3, even though it is optimal to do so.

PROPOSITION 5.2. *In a subgame perfect equilibrium of the Stochastic Replication Game, assuming that agents use tit-for-tat strategies, agents form $\frac{n}{\sigma}$ replication cliques of size σ. The cliques group agents with similar availability. If agents are numbered according to non-increasing availabilities ($u_i \leq u_{i+1}$), the j-th clique is formed by agents $\{1+\sigma(j-1), 2+\sigma(j-1), \ldots, \sigma+\sigma(j-1)\}$. The equilibrium is unique if and only if the agents' availabilities differ, i.e., $\forall i\colon u_i < u_{i+1}$.*

PROOF. The game ends when no agent changed its proposal in the next to the last round (denoted by l). As the outcome is subgame perfect, given the other agents' actions, for each agent it was *optimal* not to change any of its proposals in round l. The proof is by contradiction.

By induction, we show that agents replicate in cliques. For the first clique, assume that agent i ($1 \leq i \leq \sigma$) replicates data of at least one agent $j' > \sigma$. Thus, by tit-for-tat, at least one agent j from $\{1, \ldots, \sigma\}$ does not replicate i's data (instead replicating data of $j'' > \sigma$). Thus, i could increase its availability by stopping replication of j' ($r_l(i,0,j')$) and proposing $r_l(i,1,j)$: as $u_i < u_{j''}$ it is optimal for j to stop replicating j'' ($r_{l+1}(j,0,j'')$) and start replicating i ($r_{l+1}(j,1,i)$) (otherwise, by tit-for-tat, i would withdraw its proposal for j). This contradicts the assumption that it is optimal for i not to change its proposals in round l.

By similar reasoning, if $|R_i| < s$ (i-th storage is not fully utilized), or if $|R_j| < s$, $r_l(i,1,j)$ results in $r_{l+1}(j,1,i)$, thus both i and j gain in availability.

For the i-th clique, observe that, by the induction assumption, all the agents $\{1, \ldots, (\sigma)(i-1)\}$ replicate data between themselves, thus none of them replicates with agents $\{1+(\sigma)(i-1), \ldots, n\}$. Consequently, the same reasoning as for the first clique applies, as agents belonging to i-th clique can either replicate between themselves, or with agents with lower availability. □

The grouping corresponding to the subgame perfect equilibrium is easy to achieve in a system with centralized information in $O(n \log n)$ time. It is sufficient to sort the

agents according to non-increasing availabilities and then form replication cliques as in Proposition 5.2.

The price of anarchy quantifies the degradation of performance in the subgame perfect equilibrium compared to an equitable solution. In our system, the price of anarchy is defined as $\max_{Gj} d(G_j)_{subgame}/d(G_j)_{equitable}$. In order to compute the price of anarchy, we prove upper bounds for both fairness models (the following two propositions), and then show an instance in which the bounds are tight in the limit.

PROPOSITION 5.3. *In* $\max d(G_j)$ *aggregation, the price of anarchy is at most* $\left(\frac{\max u_i}{u_G}\right)^\sigma$.

PROOF. Value of $\max d(G_j)$ in the subgame perfect solution is determined by the group composed of the most unavailable agents that, in turn, is at most $(\max u_i)^\sigma$. In an equitable solution, $\max d(G_j) \geq u_G^\sigma$ (Proposition 4.8). □

PROPOSITION 5.4. *In* $\sum d(G_j)$ *aggregation, the price of anarchy is at most* $\left(\frac{\max u_i}{\min u_i}\right)^\sigma$.

PROOF. In any grouping (including subgame-perfect solution),

$$\sum d(G_j) = \sum \prod_{i \in G_j} u_i \leq \sum \left(\max_{i \in G_j} u_i\right)^\sigma \leq N\left(\max_{i \in \{1,\ldots,n\}} u_i\right)^\sigma.$$

The optimal grouping has (Proposition 4.9) $\sum d(G_j) \geq N\left(\min_{i \in \{1,\ldots,n\}} u_i\right)^\sigma$. □

The following proposition shows an instance in which the bounds are tight in the limit, and thus the subgame perfect equilibrium can be arbitrarily far from the equitable solution.

PROPOSITION 5.5. *In the Stochastic Replication Game, the price of anarchy is unbounded.*

PROOF. Consider an instance with $\sigma = s+1$ highly available agents (with $u_i = u_h \to 0$) and $\sigma \cdot s$ unavailable agents ($u_i = u_l \to 1$).

A equitable solution minimizing both $\sum_j d(G_j)$ and $\max_j d(G_j)$ constructs σ cliques; in each clique there is exactly one highly available agent and s less available agents (indeed, any assignment in which there are more than one highly available agent in the same clique has worse overall availability). Thus, $\sum_j d(G_j) = \sigma u_h u_l^s$; and $\max_j d(G_j) = u_h u_l^s$.

In the subgame perfect solution, highly available agents form a clique, thus leaving the less available agents to form cliques between each other. Thus, $\sum_j d(G_j) = u_h^\sigma + s u_l^\sigma$; and $\max_j d(G_j) = u_l^\sigma$.

The price of anarchy for $\min \sum d(G_j)$ is thus equal to:

$$\frac{u_h^\sigma + s u_l^\sigma}{\sigma u_h u_l^s} = \frac{u_h^s}{\sigma u_l^s} + \frac{s u_l}{\sigma u_h} \xrightarrow[u_h \to 0]{} \infty.$$

Similarly, for $\min \max d(G_j)$, the price of anarchy is equal to $\frac{u_l^\sigma}{u_h u_l^s} = \frac{u_l}{u_h} \xrightarrow[u_h \to 0]{} \infty$. □

We discuss the consequences of such a high price of anarchy in the experimental evaluation (Section 8.2).

6. AVAILABILITY-ENCOURAGING ADOPTION MECHANISM

The high price of anarchy makes the completely decentralized system almost unusable for the weakly-available agents. On the other hand, a fair, centralized solution (Section 4) does not offer incentives for agents to be highly available. In this section, we propose an alternative to these two solutions: an algorithm that uses some replication slots of the highly-available agents to help the others; but also rewards more available agents with better data availability.

6.1. Characterization of a truthful mechanism

An availability-encouraging mechanism uses some of the replication slots to make the availability distribution more fair. Agents use the remaining slots to form replication agreements as in the Stochastic Replication Game (SRG, Definition 5.1). The following game focuses on the role of the mechanism, at the same time approximating the subgame-perfect equilibrium of the SRG by approximating replicator's availability by the agent's declared availability.

Definition 6.1. The mechanism, given the declared unavailability level u'_i of each player ($0 \leq u_i \leq u'_i \leq 1$), assigns for each player i a set of replicators S_i; and for each replicator $j \in S_i$, sets the exposed unavailability level $u_j(i)$ ($u_j(i) \geq u'_j$). The pay-off of i-th player is given by:

$$d_i(u'_i, u_{-i}) = (u'_i)^{\sigma - |S_i|} \prod_{j \in S_i} u_j(i). \tag{2}$$

Both S_i and $u_j(i)$ are functions of declared unavailabilities $[u'_i]$; we omit the vector from the list of arguments in order to simplify the notation.

The notion of the exposed unavailability $u_j(i) \geq u'_j$ permits to artificially diminish the availability of the i-th agent's data stored at j: this will be crucial to design an availability-encouraging mechanism. Such a limit can be easily implemented: e.g., j can serve only a $u_j(i)/u_j$ fraction of requests for i-th data.

The fundamental problem is how to design the mechanism so that the players are motivated to declare their true availabilities, $u'_i = u_i$. The following definition formally captures that idea.

Definition 6.2. A mechanism is *availability-encouraging* (or *truthful*) if and only if for each agent i, given other agents' unavailabilities u_{-i}, the agent obtains higher or equal data availability by declaring its true availability: $d_i(u_i, u_{-i}) \leq d_i(u'_i, u_{-i})$.

Informally, an availability-encouraging mechanism "rewards" agents who increase their availability by increasing their data availability. A mechanism should be availability-encouraging, because otherwise the agents do not have an incentive to be highly available. As being available has some real costs (like electricity), in a system with an availability-*dis*couraging mechanism agents would choose low availability levels $u'_i > u_i$; thus the system would collapse.

COROLLARY 6.3. *An algorithm that forms cliques according to the subgame-perfect equilibrium (Proposition 5.2) is availability-encouraging.*

6.2. A truthful mechanism with guaranteed reduction of the price of anarchy

In this section, we propose an algorithm that builds replication agreements between agents that is truthful and has a guarantee on the reduction on the price of anarchy. The algorithm is somewhat "wasteful", in a sense that not all available replication slots are used; yet its regularity allows us to analyze its worst-case behavior. In the

subsequent section we present a heuristic that is more efficient in the average case; yet, because of the complex agreements, it is impossible to analyze theoretically.

The mechanism is implemented by a centralized algorithm that assigns to r replication slots of each highly-available agent r distinct weakly-available agents ($1 \leq r \leq \sigma$), such that each weakly available agent is "adopted by" (assigned to) k distinct agents ($1 \leq k \leq \sigma$). For instance, if $r = 1, k = 1$, the upper-half of agents adopts the lower-half; if $r = 2, k = 1$, the top 1/3 adopt the bottom 2/3. $\tau = u_{nk/(r+k)}$ denotes the unavailability of the last highly-available agent (e.g. for $r = 1, k = 1$, τ is the median of $\{u_i\}$; for $r = 2, k = 1$, τ is the first tercile). To simplify the notation, we assume that nk is divisible by $r + k$; otherwise, the last highly-available agent is $\lceil \frac{nk}{r+k} \rceil$, and its r replication slots are used to replicate agents $\{(\lceil \frac{nk}{r+k} \rceil + 1), \ldots, (\lceil \frac{nk}{r+k} \rceil + r + 1)\}$ (which are thus adopted $k + 1$ times).

The algorithm keeps a list L_W of weakly-available agents adopted less than k times (the list is initialized by $(n\frac{k}{r+k} + 1, \ldots, n)$). The list is ordered by increasing unavailabilities u. Highly-available agents are processed in order of increasing u: each highly-available agent i adopts first r agents from L_W (denoted by j), thus $S_i = \arg\min_{j \in L_W} u_j$ and $S_j = S_j \cup \{i\}$.

Next, if $k < r$, for each weakly-available agent, $r - k$ dummy agents with $u_d = 1$ are added to the replicas set S_j. These $r - k$ replications slots are wasted: as we demonstrate later, we need this for the truthfulness of the mechanism; at the same time, these slots do not play a role in the reduction of the price of anarchy. A heuristic proposed in the next section uses these slots more effectively (but has no asymptotic guarantee on the price of anarchy).

Finally, the exposed unavailabilities are set. Adopted agents j expose to their partners the maximal unavailability u_n. This bounds the incentive for highly-available agents to manipulate the threshold. Alternatively, if both r and k is fixed (as in Proposition 6.4), an adopted agent j can expose to its parent its true unavailability $u_j(i) = u_j$. As the highly-available agents are processed in order of increasing u, this is sufficient to guarantee that the data unavailability increases with agent index.

In contrast, for the weakly-available agents, the mechanism must ensure that the resulting data unavailability increases with agent index; thus $u_i(j)$ are set so that $d_j \geq d_{j-1}$. Approximating d as in Definition 6.2, $u_i(j)$ are derived from $d_j = \max(d_{j-1}, u_j^{\sigma-r} \prod_{i \in S_j} u_i(j)$.

The following proposition shows that the algorithm is truthful if r and k are chosen independently of the distribution of unavailabilities u. For instance, the system might announce r and k, and then gather the agents' declarations of u.

PROPOSITION 6.4. *Given r, k, the mechanism is truthful.*

PROOF. We consider three cases: (i) a highly-available agent ($u_i < \tau$) declaring $\tau \geq u'_i > u_i$; (ii) a highly-available agent declaring $u'_i > \tau$; (iii) a weakly-available agent ($u_j > \tau$) declaring $u'_j > u_j$.

In the first case, $d_i(u'_i, u_{-i}) = (u'_i)^{\sigma-r} \prod_{j \in S'_i} u_j(i) > (u_i)^{\sigma-r} \prod_{j \in S_i} u_j(i)$, as $u'_i > u_i$ and a less-available agent can have less available partners assigned by the mechanism: as highly-available agents are processed by decreasing u_i, some agents from S_i might be not longer available and replaced by weaker agents in S'_i, thus $\prod_{j \in S'_i} u_j(i) \geq \prod_{j \in S_i} u_j(i)$.

In the second case, as $u'_i > \tau$, and the exposed unavailabilities are set so that d'_i is increasing, $d'_i \, d'_i \geq d_i$.

In the third case, by declaring $u'_j > u_j$, the agent can lower its ranking if and only if there exist an agent l with $u_j < u_l < u'_j$. As d is set to be increasing, $d'_j \geq d_l \geq d_j$. □

The following proposition shows that the reduction of the price of anarchy is proportional to the threshold τ^k.

PROPOSITION 6.5. *The mechanism reduces the price of anarchy by at least* $\tau^k / u_n^{\max(r,k)}$ *if* $r + k \leq \sigma$.

PROOF. The price of anarchy is determined by the data unavailability $d(n)$ of the least available agent n. Without the mechanism, the price of anarchy is bounded by $\left(\frac{u_n}{u_G}\right)^\sigma$ (in $\max d(G_j)$ aggregation, Proposition 5.3) and by $\left(\frac{u_n}{u_1}\right)^\sigma$ (in $\sum d(G_j)$ aggregation, Proposition 5.4).

After the mechanism completes, all weakly-available agents have k partners assigned with unavailability of at most τ. Thus, before limiting the exposed unavailabilities (the last step of the algorithm), the weakly-available agents had the data unavailability at most $d_j \leq u_j^{\sigma-\max(r,k)}\tau^k$. The least-available highly-available agent (with index $p = nk/(r+k)$) has its data availability of at least $d_p \leq \tau^{\sigma-r}u_n^r$. If $r+k \leq \sigma$, $\tau^{\sigma-r}u_n^r \leq \tau^k u_n^{\sigma-\max(r,k)}$ Thus, after limiting the exposed unavailability, the least available agent n has still the data unavailability $d_n \leq u_n^{\sigma-\max(r,k)}\tau^k$. □

We treat r as given, as it directly translates into burden for the highly-available agents. In contrast, k should be optimized by the algorithm. Given vector of agents' unavailabilities u ($u[i] = u_i$), as the reduction in the price of anarchy is proportional to $\tau^k = (u(n\frac{k}{r+k}))^k$, the algorithm should choose an optimal k^* i.e., $k^* = \arg\min_{1\leq k\leq\sigma-r} u(n\frac{k}{r+k})^k$.

However, as k^* depends on the declared unavailabilities u, some agents might be interested to declare $u'_i > u_i$ in order to manipulate this choice. Smaller values of k result in smaller thresholds τ, thus more agents are treated as weakly-available and have assigned partners. In general, for the same declared unavailability u_i, an agent i prefers to be treated as weakly-available than as highly-available.

We denote by p^* the agent that defines the threshold τ^* for the optimal k^* (assuming that every agent declares its true unavailability), i.e., $p^* = n\frac{k^*}{r+k^*}$; and by p' the agent that defines the threshold τ' for $k' < k^*$, i.e., $p' = n\frac{k'}{r+k'}$. Agents do not have incentive to manipulate the threshold so that $k' > k^*$, as agents that become highly-available $\tau^* \leq u_i \leq \tau'$ can be worse-off than before; and agents that are highly-available for both thresholds ($u_i \leq \tau^*$) do not gain from this change as the unavailability exposed by the weakly-available agents is u_n.

Assume that $p' < i \leq p^*$ (as other agents have no incentive to change the threshold). If k' is chosen, the agent i is treated as weakly-available as $\tau' < u_i$; its $d'_i = (u'_i)^{\sigma-k}(\tau')^k$. If k^* is chosen, the agent i is treated as highly-available; its $d_i = (u_i)^{\sigma-r}u_n^r$. Thus, it is possible that the agent improves its data availability by being treated as a weakly-available: for some k, r, τ', u_i, $d'_i < d_i$.

Given a vector of other agents' unavailabilities u_-, how can an agent i influence the mechanism so that k' rather than k^* is chosen? If i declares $u'_i = u(p^*+1)$, the p^*-th agent in the original vector u (that determined the optimal threshold τ^*) becomes the (p^*-1)-th agent in the resulting distribution u'. Thus, the threshold is determined by the (p^*+1)-th agent in the original distribution u. (declaring $u'_i > u(p^*+1)$ is not efficient for i, as it does not further influence the threshold, and, once the threshold is set, the agent is better-off declaring u'_i as low as possible by Proposition 6.4).

If declaring $u'_i = u(p^*+1)$ is sufficient for the mechanism to switch from k^* to k', how does it affect the reduction of the price of anarchy? As the mechanism switched from k^* to k' for $u = (u'_i, u_-)$, $u_{p'}^{k'} \leq u_{p^*+1}^{k^*}$. By dividing this inequality by $u_{p^*}^{k^*}$, we

get $u_{p'}^{k'}/u_{p^*}^{k^*} \leq u_{p^*+1}^{k^*}/u_{p^*}^{k^*}$. Thus, the relative loss of the performance of the mechanism $u_{p'}^{k'}/u_{p^*}^{k^*}$ is at most equal to the relative difference of unavailabilities of two consecutive agents $\left(u_{p^*+1}/u_{p^*}\right)^{k^*}$.

Of course, if $k' > 1$, other agents might have an incentive to lower the threshold even further. However, there might be at most k^* such actions. We denote by p_l the agent defining threshold τ_l ($1 \leq l \leq k^*$), the cumulative impact on the mechanism is at most $\prod_{1 \leq l \leq k^*} \left(u_{pl+1}/u_{pl}\right)^{k^*} \leq \left(\max_{1 \leq l \leq k^*}(u_{pl+1}/u_{pl}\right)^{k^* \cdot k^*}$

As the distribution of agents' unavailabilities u is measured, we may assume that it is "dense" in a sense that the relative difference between two consecutive agents is small. Consequently, even if a mechanism optimizing k is not strictly truthful, the possible manipulations are small; and its result does not significantly impact the price of anarchy. Moreover, the game considered in this section is a worst-case approximation of what happens in a real system. In the real system, the unavailability u_i is measured, and not declared by the agents. The impact of the measurement tool can be modelled as a random noise added to the unavailability declared by an agent; thus, the impact of slight, strategical increase of unavailability (as analyzed in this section), can be cancelled by a random noise; and, in general, a significant increase of unavailability is not profitable by Proposition 6.4.

6.3. A Heuristic Adoption Mechanism

The mechanism proposed in this section does not waste the replication slots compared to the theoretical mechanism presented previously. However, because of the complex graph of replication agreements, we will not derive a closed formula for the reduction of the price of anarchy.

The mechanism is still truthful in the sense of Proposition 6.4, i.e., once the parameters are set, the algorithm guarantees that the data unavailability is increasing with agents' unavailabilities.

Also, similarly to the previous section, the algorithm takes as a parameter the maximum number of slots r it can use at each agent. Unlike the previous algorithm, however, it can use less than r slots.

The algorithm (see Algorithm 1) does a binary search for the optimal target unavailability t^*. For each tested value of t, the algorithm tries to organize the replication agreements so that all the agents have data unavailability at most t, $d(i) \leq t$. If the assignment fails, it means that the tested value was too low; if it succeeds, it tries a lower t.

Given t, the algorithm sets r boundary thresholds τ_k to $\tau_k = t^{1/(\sigma-k)}$ ($k \in \{1, \ldots, r\}$). Agents with $\tau_{k+1} < u_i \leq \tau_k$ contribute exactly k slots to adopt weakly-available agents.

Then, weakly-available agents ($u_j > \tau_1$) are processed in order of increasing availabilities (from the least available to the most available). For each agent j, the algorithm greedily assigns highly-available replication partners to achieve $d_j \leq t$ in the following loop. If there is a single highly-available agent i sufficient to achieve $d_j \leq t$, then the worst-possible i is assigned and the loop breaks. Otherwise, the highest currently available i is assigned to j and the loop continues. If all highly-available agents have all their adoption slots assigned, the subroutine exits with failure (t was too low). After assigning replication partners for j, the algorithm tries to greedily swap j's best replication partner $p_m = \min\{i \in S_j\}$ with a weaker-available agent i. i is chosen so that if p_m is replaced in the replication group by i, j's data unavailability is still below the threshold t, $p' = \max\left\{p : d_j(S_j - \{p_m\} \cup \{p\}) \leq t\right\}$.

If the algorithm succeeds to assign replicators to all weakly-available agents, the exposed unavailabilities $u_i(j)$ are bounded in order to keep d_i increasing with i (in the

```
t_min = 0; t_max = 1
while t_max − t_min > ϵ do
    t = (t_max − t_min)/2
    for k = 1 to r do
        τ[k] = t^{1/(σ−k)}
    for i = n downto 1 do
        k = 1; S_i = ∅
        while u_i < τ[k] do
            k++
        r_i = k − 1
        if r_i = 0 then
            p = i
    for j = n downto p do                          ▷ p denotes the most-available child
        d = u_j^σ; failed=false
        while d > t and not failed do
            failed=true
            for i = p − 1 downto 1 do
                if j ∉ S_i and |S_i| < r_i and d · u_i/u_j < t then
                    S_j := S_j ∪ {i}; S_i := S_i ∪ {j}; d := d · u_i/u_j
                    failed=false; break
            if d > t then
                for i = 1 to p do
                    if j ∉ S_i and |S_i| < r_i then
                        S_j := S_j ∪ {i}; S_i := S_i ∪ {j}; d := d · u_i/u_j
                        failed=false; break
        if failed then
            break
        p_m = min{i ∈ S_j}; d = d/u_pm
        for i = p − 1 to p_m do
            if j ∉ S_i and |S_i| < r and d · u_i < t then
                S_j := S_j ∪ {i} − {p_m}; S_i := S_i ∪ {j}
                S_pm := S_pm − {j}
    if failed then t_max := t else t_min := t
```

Algorithm 1: A heuristic adoption mechanism.

same way as in Section 6.2; this last step is not shown in Algorithm 1). Similarly to Proposition 6.4, by the definition of the truthful mechanism 6.2, this final step makes the mechanism truthful.

7. A HEURISTICS FOR THE DECENTRALIZED MATCHING

The following algorithm creates an environment similar to the Stochastic Replication Game (Definition 5.1). The main goal of the algorithm is to reduce the time needed to reach the subgame-perfect equilibrium (Proposition 5.2) in the context of a real distributed system, that, through limited bandwidth and agents' processing power, limits the number of replication proposals that each agent can make. Among replication candidates (most of whom are unknown due to the distributed nature of the system), the algorithm helps agents find and choose partners that not only maximize data availability, but also are not likely to withdraw replication agreements—thus, the partners from the equilibrium. At the same time, all the decisions imposed by the algorithm are rational (never decrease the agent's data availability), thus the algorithm converges to cliques defined in the subgame perfect equilibrium. Consequently, even if some of the agents choose not to follow the algorithm (but still are rational), the steady state will be the same—the equilibrium—but reached more slowly (or faster, if the aberrants use, e.g., an oracle).

To illustrate this difference, consider an agent with a medium availability. To maximize its data availability, the agent should try to form a replication agreement with

function $score_c$(agent i, agent j, $G_i : i \in G_i$, $G_j : j \in G_j$)
  if $|G_i| < \sigma$ **then**
    if $|G_i| + |G_j| \leq \sigma$ **then**
      return $|av_i - av_j|$
    else if $|G_j| < \sigma$ **then**
      return $0.5 \cdot |av_i - av_j|$
    else
      return 0
  else
    $u_i^H = \max_{i' \in G_i} u_{i'}$; $u_i^L = \min_{i' \in G_i} u_{i'}$
    $u_j^H = \max_{j' \in G_j} u_{j'}$; $u_j^L = \min_{j' \in G_j} u_{j'}$
    if $u_j^H < u_i^L$ or ($|G_j| = \sigma$ and $u_j^L > u_i^H$) **then**
      return 0
    else
      return $\max(u_i^H, u_j^H) - \min(u_i^L, u_j^L)$

Algorithm 2: Agent i computes the score of agent j using Explicit Cliques.

an agent with high availability. However, such a highly available agent is likely to already have (or have in the near future) highly available replication partners; thus the replication request from the "mediocre" agent will be either rejected, or withdrawn soon.

Each agent maintains a list of candidates for replicators. This list is refreshed by the T-Man [Jelasity and Babaoglu 2006] gossiping protocol. Each agent i has two pools of agents: a random pool $rand(i)$—at most s_r agents forming a sample of the population; and a *metric* pool $metric(i)$ with s_m agents that score well according to a local metric. In an iteration of T-Man, each agent updates its random pool by gossiping with a randomly-chosen agent from this pool. During this operation, to form the new random pool, each agent chooses s_r most recently added agents from both random pools. After modifying the random pool, the metric pool is updated as the best s_m agents from the current metric pool and the current random pool. Then, the agent communicates with the best agent from its metric pool: the metric pools are exchanged, merged, scored and then each agent chooses the best s_m agents from the merged pools.

To form replication agreements, agents use heuristics to compare the current replicators with the candidates. We first describe a framework, then several possible heuristics to choose candidates.

In each turn, each agent i scores the agents in its metric pool that are not i's current replicators nor on its taboo list. If i has less replicators than its maximum capacity, it proceeds to querying the agent j^* with the highest score. Otherwise, candidates are compared with i's worst current replicator l ($l = \arg\max_{m:\, replica(i,m)} u_m$). i queries the first candidate j^* (in order of non-decreasing score) better than l (for which $u_j < u_l$). If there is no such candidate, i does not switch replicas.

The queried candidate j^* decides whether to accept the mutual replication: if it has less replicators than its maximum capacity, i is accepted. Otherwise, i is accepted only if $u_i < u_{l'}$, where l' is j^* worst replicator.

Finally, if j^* accepts i, and i already has as many replicators as its maximum capacity, i drops its current worst replicator l.

In order not to repetitively query the same agents, each agent i maintains a taboo list, consisting of former replicators that have been dropped or that dropped i; and of agents that did not accept replication with i.

We used three variants of the above algorithm that differ in the trade-off between short-sighted selfishness and the speed of convergence (we compare these algorithms in Section 8.7). In *Optimistic Queries*, candidate j's score is equal to its availability $score_o(i, j) = a_j$. In *Pragmatic Queries*, candidate j's score is equal to the abso-

lute difference between its availability and the availability of the assessing agent i, $score_P(i,j) = 1 - |a_i - a_j|$.

Finally, ***Explicit Cliques*** maintains cliques composed of one or more agents. Every member of a clique replicates data of all other members. Thus, a representative i of a clique G_i, after choosing agent j^* with the maximum score $score_C(i,j)$, tries to merge its clique with the clique $G_j : j^* \in G_j$ of the chosen candidate. The two cliques exchange members: the "better" clique groups σ agents with the highest availability (or the two cliques combined, if the combined clique has at most σ members); the "worse" clique groups remaining members of both cliques.

The scoring function $score_C(i,j)$ depends on the size of the cliques $i \in G_i$ and $j \in G_j$ (see Algorithm 2).

If $|G_i| < \sigma$, G_i is not complete and the algorithm should increase its size (as it considerably reduces $d(G_i)$). It is best to have G_j such that the two cliques will be merged into one, $|G_i|+|G_j| \leq \sigma$. Thus, $score_C(i,j) = score_P(i,j)$ in this case; otherwise, if $|G_j| < \sigma$, $score_C(i,j) = 0.5 score_P(i,j)$; finally if $|G_j| = \sigma$, $score_C(i,j) = 0$ (as in this case one of the cliques after merging will still be of size $|G_i|$).

If $|G_i| = \sigma$, the goal is to find a clique G_j that after merging will reduce the variance of availabilities of agents in both cliques. During merging of G_i with G_j, the two cliques will exchange members (and thus reduce the variance) if: (i) the intersection of availability ranges is not empty, $[u_i^L, u_i^H] \cap [u_j^L, u_j^H] \neq \emptyset$; or (ii) $|G_j| < \sigma$ and $u_j^H > u_i^L$. In these two cases, the score is equal to the range of unavailability that will be reduced; otherwise $score_C(i,j) = 0$.

8. SIMULATION OF THE ALGORITHMS

In this section we assess the performance of the matching algorithms by simulation on a realistic distribution of peers' availabilities. We consider both centralized and decentralized matching. Using centralized matching algorithms, we experimentally verify that the high price of anarchy makes a storage system unusable for the newcomers (and weakly-available agents). We also show that the adoption mechanism results in acceptable data availability for the weakly-available agents. We repeat these experiments on a population with a fraction of agents renting infrastructure from cloud storage providers, verifying that the cloud has a significant, positive impact on the system. We then study the decentralized matching algorithm, showing that the subgame perfect equilibrium can be efficiently computed by a decentralized algorithm even under churn.

8.1. Simulation Settings

As we consider a snapshop of the system, the only input to our simulation is the distribution of agents' availabilities (except analyzing churn in Section 8.7). Excluding the cloud-assisted configuration (Section 8.4), and the imprecise estimates (Section 8.6), agents' availabilities were generated in three steps. Firstly, following [Bernard and Le Fessant 2009], 10% of the agents have availability 0.95, 25%—0.87, 30%—0.75 and 35%—0.33. Then, we added a Gaussian noise with standard deviation of 0.1 to each availability. Finally, we capped the resulting value, so that $0.03 \leq a_i \leq 0.97$. Histogram on Figure 1 shows the resulting distribution of agents.

We generated agents' availabilities with a stochastic model, rather than used a trace, for a few reasons. First, using a stochastic model enables us to repeat experiments and thus analyze statistical significance, rather than anecdotic behavior. Second (rephrasing some of the arguments of [Bernard and Le Fessant 2009]), we are not aware of any publicly-available trace of either a distributed storage software or that exhibits behavior that is comparable to our scenario. A vast majority of traces describe p2p file

sharing systems. File-sharing has a different use-case than distributed storage: in file sharing, once an agent downloads a file, it has no further incentive to share it (perhaps except BitTorrent private trackers, but, again, we are not aware of any publicly-available trace from such a tracker). We envision that, as our system incentives agents to be available, our software will be run whenever the agent's machine is on-line. The Microsoft desktop PCs trace [Bolosky et al. 2000] (which we used in Section 8.6) shows machines availability, which makes it the most applicable to our setting; however, the trace is from a single, corporate environment and, moreover, 15-years old (thus, it misses or under-states, e.g., notebooks, mobile devices, or smart routers).

We repeated each experiment on 50 instances with agents' availabilities generated as described above; error bars on plots denote standard deviations. As the error bars can make lines in the figures similar, legends describe lines in the same order as the lines appear on the left side of figures.

We set the storage size $s = 5$ and the sizes of random and metric pools in T-Man gossiping to 50.

We implemented *decentralized* algorithms in a custom discrete event simulator. We simulate the organizational phase of the system; no data is transmitted until the solution stabilizes. In each round of the simulated matching, all the agents are processed sequentially in random order. Each agent performs one iteration of T-Man gossiping, and then one iteration of the decentralized matching algorithm.

8.2. Centralized Allocation: Subgame Perfect vs Equitable Solutions

We started with comparing *random*, *subgame perfect* and *equitable* allocation algorithms according to the resulting data unavailability. The random allocation corresponds to unoptimized replication, as, for instance, in a DHT, Pastiche [Cox et al. 2002], or Total Recall [Bhagwan et al. 2004]; or optimized for a different criteria than availability [Giroire et al. 2009]. Equitable allocation is analogous to a global optimization of file availability, as, for instance, in Farsite [Douceur and Wattenhofer 2001a; 2001b]. We ran these algorithms on instances of 10.000 agents each; then we computed averages over all the random instances and all agents having similar availabilities (with resolution equal to two decimal places, e.g., the score for 0.95 is an average for all agents with availability in range $[0.95, 0.96)$). Figure 1 summarizes the obtained results.

The equitable algorithm produces cliques that result in similar data availability regardless of the agent's availability. In contrast, the subgame perfect equilibrium results in wide range of data availabilities: while the highly available agents have their data available with expected failure probability of approximately 10^{-9}, the weakest available agents almost do not gain from replication, with data unavailability close to 1.

Such diversification in the subgame perfect solution provides incentives for agents to be highly available. A highly available agent is able to replicate its data with other highly available agents, which exponentially increases the agent's data availability. However, the subgame perfect solution might be too "extreme" to the less-available agents. Agents with availabilities less than approximately 0.5 have their data available with probability less than 0.99 (approximately), which might be not sufficient for some applications. This, in turn, can discourage such agents from joining the system, and consequently, prohibit the system from growing to a critical mass.

On the other hand, an equitable solution does not reward highly available agents. In absence of altruistic agents, the system would degenerate.

Also note that the equitable solution clearly Pareto-dominates the random assignment, resulting in higher data availabilities for all classes of agents.

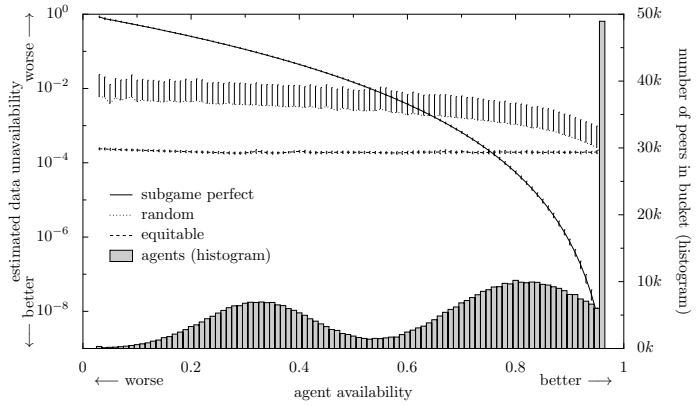

Fig. 1. Agents' expected data unavailability as a function of their availability in random, equitable and subgame perfect assignment. Histogram shows the number of agents in each availability bucket.

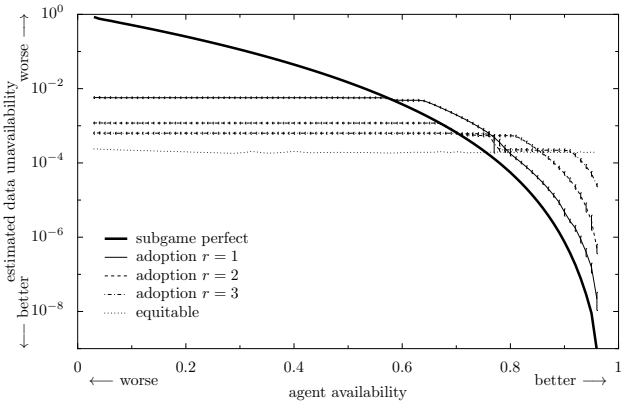

Fig. 2. Agents' expected data unavailability as a function of their availability in adoption-based mechanism for different number of adoption slots r. Subgame-perfect and equitable assignments given as a reference.

8.3. Centralized Allocation: Increasing Availability by Adoption

We simulated the influence of the availability-encouraging adoption (Section 6) on the data availability. We varied the number of adoption slots $r \in \{1, 2, 3\}$ and, for each r and each distribution of agents' u_i, performed the heuristic algorithm from Section 6.3. The remaining slots were assigned by a subgame-perfect solution. The rest of the settings for the experiment was the same as in the last experiment. Figure 2 shows the results.

The adoption mechanism is efficient in increasing the data availability d of the weakest-available agents. Even with one slot donated by the highly-available agents ($r = 1$), the data availability d of the 20 weakest-available agents is improved by, on the average, two orders of magnitude (128.4 times, std.dev. 7.9). Agents with availabilities smaller than 0.6 have greater data availability than in the selfish solution. All weakly available agents have similar data availability, thus the mechanism does not "reward" agents for increasing availability up to, approximately, 0.65. The mechanism is not, however, discouraging agents to declare their true availability, as the curve is non-increasing. If one wants to emphasize rewards, the mechanism can be easily modified to decrease the data availability with decreasing availability: the target unavailability t should be decreased with agent index. The impact of the mechanism on

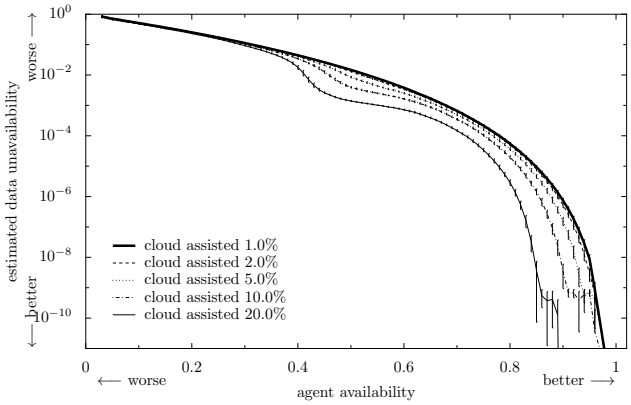

Fig. 3. Agents' expected data unavailability as a function of their availability in the subgame perfect assignment. Lines denote different proportion of peers using a cloud provider with .9999 availability.

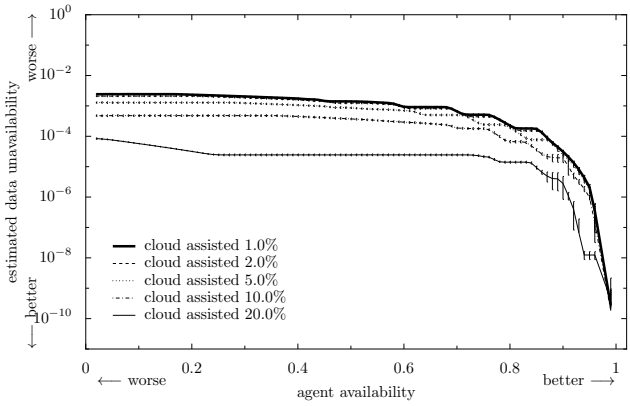

Fig. 4. Agents' expected data unavailability as a function of their availability in the adoption-based mechanism for $r = 2$ adoption slots. Lines denote different proportion of peers using a cloud provider with .9999 availability.

highly-available agents is limited: their data availability is decreased by about two orders of magnitude (corresponding to approximately one replication slot "wasted" by the mechanism on these agents). This behavior is caused by the fact that the highest-available agents are assigned the weakest-available partners.

Donating more replication slots results in even higher gains for the weakly-available agents. For $r = 2$, the data availability is improved on the average 618 times (std.dev. 39); for $r = 3$, by more than three orders of magnitude (1159 times, std.dev. 79).

The threshold τ found by the algorithm decreases with increasing r. For $r = 1$, $\tau_1 = 0.35$; $r = 2$, $\tau_1 = 0.26$, $r = 3$, $\tau_1 = 0.23$ (std. dev. less than 0.005; τ denotes the *unavailability* of the weakest parent). For $r = 1$, approximately 40% of the population is treated as weakly-available; this share grows up to 57% for $r = 3$.

8.4. Centralized Allocation: Cloud-Assisted System

As we envision that some of the users, rather than providing physical resources, will run the software on resources rented from IaaS (Infrastructure as a Service) providers, we wanted to verify their impact on the system. We set the proportion of these cloud agents in different experiments to 1%, 2%, 5%, 10% and 20%. Each cloud agent rents

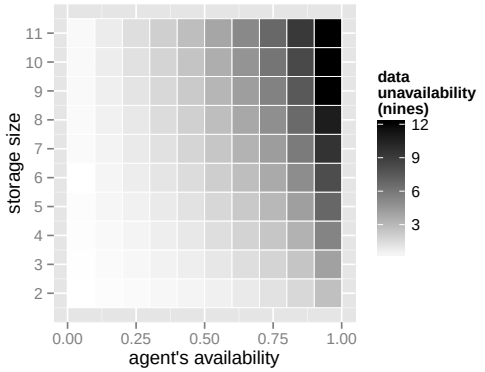

Fig. 5. Agents' expected data unavailability as a function of their availability and the storage size in the subgame perfect assignment. Shades of gray denote data unavailability measured in nines: e.g., black corresponds to $d_i = 10^{-12}$.

its infrastructure from a certain provider; agents using the same provider cannot form a replication agreement (not to introduce correlated failures). [SA 2013] reports that the top 5 cloud storage providers have market shares of, respectively 27%, 17%, 15%, 10% and 4% (and 55% of users do not use the cloud storage); we thus limit the number of providers in our experiment to 4 (as the fifth provider is more than twice smaller than the forth), and set their relative market shares accordingly (39%, 24%, 21%, 14%). The availability of a cloud agent is set to 0.9999 (Amazon S3 SLA, [AmazonS3 2013]). The results are shown in Figures 3 and 4.

The results demonstrate that the whole system gains by incorporating highly-available cloud-assisted agents. As the market shares of cloud providers are not equal, agents from the popular provider have not enough partners in other providers. Thus, even in the subgame perfect equilibrium (Figure 3) a significant portion of cloud agents replicate non-cloud agents. When the adoption mechanism is used (Figure 4), the cloud agents enable others to achieve excellent data availability: with 10% of the cloud agents, the resulting data unavailability of the worst-off agents is around 0.0002.

8.5. Centralized Allocation: Heterogeneous Storage

In the subgame perfect assignment, if there are many agents, agent's expected data unavailability corresponds roughly to its unavailability to the power of its storage size, $d_i = u_i^{s_i+1}$. Thus, a weakly-available agent, if it cannot improve its availability, may instead decide to increase its storage size—compensating the quality of its replicators by their quantity.

We simulated a system having the same distribution of agents' availabilities, but we set each agent's storage size s_i to a value from uniform distribution over $[2, 11]$ (thus, an agent has at least one replicator, and at most 10). When summarizing results, to improve presentation, we capped d_i to 10^{-12} (as other effects, such as network failures, would dominate agents' unavailabilities). Figure 5 shows the resulting data unavailabilities.

The results show that agents can indeed use larger storage size to achieve better data availability. For instance, agents with availability $a_i = 0.25$ and storage size $s_i = 11$ have similar data unavailability as agents with $a_i = 0.45$ and $s_i = 6$; or $a_i = 0.65$ and $s_i = 3$. This results suggest an alternative method to make the storage system reasonable to newly connected agents.

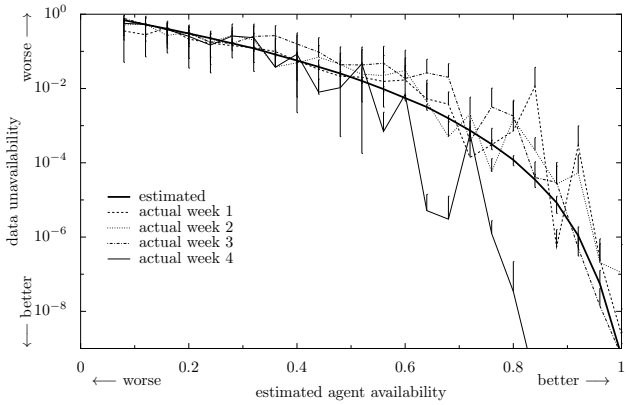

Fig. 6. Estimated (bold black line) and true data unavailability for different weeks in subgame-perfect equilibrium. To improve readability, agents are grouped in 25 buckets. Skype superpeers trace.

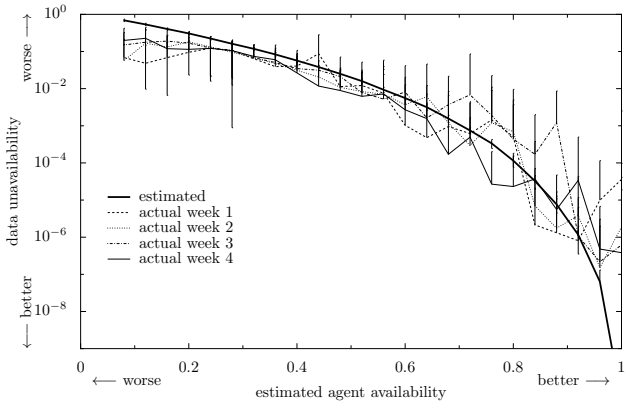

Fig. 7. Estimated (bold black line) and true data unavailability for different weeks in subgame-perfect equilibrium. To improve readability, agents are grouped in 25 buckets. Microsoft workstations trace.

8.6. Centralized Allocation: Imprecise Estimates of Agents' Availabilities

Our data availability model assumes that agents' availabilities a_i estimate agents' future availability. In this series of experiments, we investigate how the results change when availability estimates are imprecise, i.e., the observed availability differs from the system-generated estimate.

In order not to propose a probabilistic model of agents' behavior, in this series of experiments we used two published traces: skype superpeers [Guha et al. 2006] and Microsoft desktop PCs [Bolosky et al. 2000]. We chose these traces as, in our opinion, the machines' availability patterns might exhibit similar characteristics to a storage system. However, as we argue in the introduction to the experimental section, the users' behavior described in these traces do not necessarily match a modern storage system.

Both traces report machines' online periods during about a month. We divided the trace into 4 periods of one week. We used a straightforward estimator of future availability: for each machine i, we assumed that the estimated availability a_i^t in week $t > 1$ is equal to this machine's average (true) availability a_i^{t*} in weeks $1, \ldots, (t-1)$ (a_i^{t*} is measured as the fraction of time the machine is online). We chose the average as es-

timator for its simplicity. For a particular system or when traces are available, more sophisticated statistical or machine learning methods can be used to construct a more precise model of the agent's availability. We used the estimator a_i^t to construct the replication groups; then we calculated the true average data unavailability in week t using the actual machine availability in that week a_i^{t*}. When converting traces, we set 0.9999 (Amazon SLA) as the maximal machine availability. Figures 6 and 7 summarize the obtained results (note that the estimated data unavailabilities for all weeks are similar, thus we plot just a single estimate).

For both traces, the true data unavailabilities follow the general distribution of the expected data unavailabilities—weakly-available agents have significantly lower data availability than highly-available agents. However, the variability of the data unavailability is high (as standard deviation lines show). Additionally, the two traces differ in how the estimates correspond to the actual values: in the Microsoft trace, low availabilities are under-estimated and high availabilities over-estimated; whereas in the Skype trace low availabilities are estimated more precisely and high availabilities are estimated less precisely.

The results suggest that estimating availabilities is an important problem for a distributed storage system. First, the traces might be too short to get a true "average" estimate. Indeed, when wuala.com allowed users to "trade" their storage space, the user had to have at least a month's history of stable connectivity. Second, our estimate was just average availability: other estimates [Mickens and Noble 2006] might be more precise (although this is out of the scope of this paper).

8.7. Decentralized Allocation: Speed of Convergence

In the next series of experiments, we measure how fast do the decentralized algorithms presented in Section 7 converge to the subgame perfect cliques.

All algorithms use a taboo list not to repeatedly try a replication agreement with the same replication partners. *Optimistic Queries* should not delete an agent from a taboo list, because it is almost certain that the agent that refused a replication agreement will not change its decision. However, *Pragmatic Queries* and *Explicit Cliques* might gain from limiting the number of agents on the taboo list (to e.g. $2s$, as the optimal replication agreements are formed by s among $2s$ neighbors). In the initial experiments, we checked that such a limit does not influence the convergence speed of *Explicit Cliques*; and that *Pragmatic Queries* converges slower when the size of the taboo list is limited. If a list is unlimited, there is a small risk that an optimal replication partner is added to the taboo list; but, according to our experiments, this risk is outweighed by the gain an agent gets from reducing redundant queries.

Initial experiments revealed that the *Optimistic Queries* version of the algorithm is inefficient. After the first few rounds when the underlying gossiping protocol efficiently fills the metric pools of all agents with the same set of 50 highest available agents, in the subsequent rounds the whole population queries the best agent, the second-best agent, and so on. Thus, replication agreements are formed extremely slowly. We observe that if agents' availabilities are distinct, approximately R/σ cliques are formed after approximately R rounds.

Figure 8 compares the convergence speed of *Pragmatic Queries* to *Explicit Cliques*, measured as the median average degradation vs. the subgame perfect solution (we treated differences less than 10^{-9} as zero). In this experiments, we excluded agents with boundary availabilities (0.97 and 0.03) from the results.

Explicit Cliques converges much faster (in about 25 rounds), by quickly building as many full cliques as possible, and then optimizing their contents. High standard deviation observed in rounds 18-25 is an artifact of computing the median from only few values.

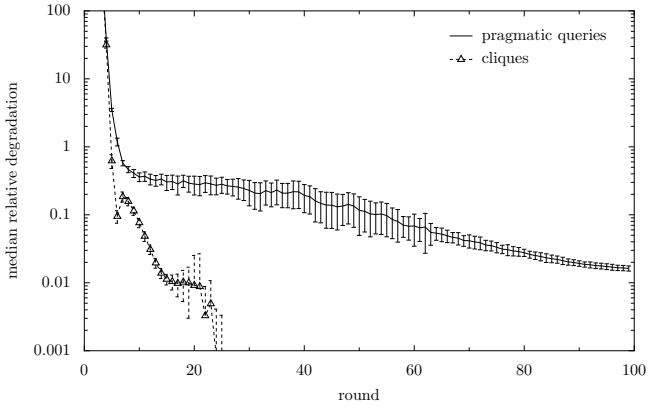

Fig. 8. Convergence speed of decentralized algorithms to subgame perfect cliques. Y axis is the median (computed over only non-zero elements) of relative degradation vs. the subgame perfect solution. $n = 2000$

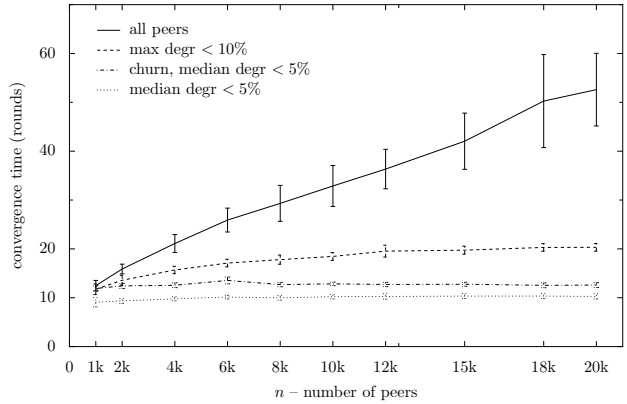

Fig. 9. Convergence speed of the clique-based algorithm to the subgame perfect cliques as a function of the number of agents in the system. The solid line denotes the number of rounds needed to reduce the degradation of all agents to less than 10^{-9}; the dashed—to reduce the maximum relative degradation to less than 10%; the dotted—to reduce the median degradation to less than 5%.

In contrast, *Pragmatic Queries* form two chains of agents (grouping highly available agents in one and weakly available agents in the other). The chains are formed because each agent i replicates with s closest neighbors according to the absolute value of the difference in availabilities: $s/2$ agents with availabilities higher than i and $s/2$ agents with lower availabilities. Only the $s/2$ most-available agents, lacking even higher available agents, form agreements with worse agents. In next rounds, these worse agents gradually drop their chain neighbors in favor of higher available agents; thus, the dropped neighbors no longer have higher available neighbors, and the phenomenon propagates towards the next agents.

Figure 9 shows the speed of convergence of *Explicit Cliques* as a function of number of agents. The algorithm reduces the median degradation to less than 5% in about 10 rounds; that result does not depend on the size of the population. Even for large populations, it takes just around 20 to reduce the maximum degradation to less than 10%. In contrast, the number of rounds for the absolute convergence depends on the size of the population: the mean number of rounds rises from 12.5 (for 1,000 agents) to

52.6 (for 20,000 agents). For the largest population, the algorithm converged in at most 71 rounds.

We also simulated the impact of temporal failures when the algorithm is running. To simulate the worst-case behavior, we assumed that once an agent leaves, it does not return to the system; and there are no new agents joining. In order to model the probabilistic process of leaves, we needed to characterize the duration of the distributed algorithm. Each agent during one round of the algorithm contacts a single agent from its random pool and a single agent from its metric pool; as the amount of changed information is negligible, we may bound the length of that phase by two maximal round trip times (RTTs) in the network. We upper-bounded the duration of a round to 1 second. The process of agents' leaves depends directly on expected sessions' durations. A storage system would typically have long sessions; to simulate the worst-case behavior, we assumed that the mean session length is just 1,000 seconds (roughly 17 minutes). This leads to a probabilistic model in which in each round each agent has a probability of $1/1000$ of leaving the system.

We measured the number of rounds needed to reduce the median degradation to less than 5%. Certainly, measuring the absolute convergence does not make sense, as if an agent leaves, its replication partners with higher availability would have their data availability degraded (as they need to choose worse replication partners). The results are summarized by a dash-dotted line on Figure 9. Compared to a no-churn system, it takes 2-3 rounds more for the system to converge. The algorithm is thus robust in the presence of high churn.

9. DISCUSSION: HOW REALISTIC ARE OUR RESULTS

In this paper, we deliberately focus on a single issue, the replica placement, which allows us to derive mathematical, as well as simulation results. We leave unaddressed other problems like maintenance [Chun et al. 2006; Yang et al. 2011], redundancy schemes [Rodrigues and Liskov 2005], or heterogeneity of agents' storage needs and capabilities.

We assume that the sole parameter describing an agent is its availability; in large-scale networks an agent judging a replication partner may also take into account other factors, such as the estimated bandwidth of the connection (see e.g. [Liu and Datta 2012]). Such estimates are subjective: an agent can be ranked high by agents who are "close" in terms of network bandwidth; whereas other agents that need to pass through e.g. a congested long-range link, would rank the agent low. In our future work, we plan to address replication agreements with subjective rankings. However, our results hold inside each network region; thus, an agent may pre-filter potential replication partners based on the bandwidth and then apply our algorithm.

In the theoretical analysis, we consider that availabilities are probabilistic estimates of the future behavior of a peer; these estimates do not depend on time. Our earlier work [Rzadca et al. 2010] proposed and analyzed also another model, in which estimates describe the availability depending on time of day or week-day (this approach can model e.g. daily and weekly usage patterns of a computer). Our initial results suggested that time-based analysis improves availability only when a system has a truly global scope: downtimes of agents from one continent (e.g. Europe) can be complemented by others (e.g. Asia). In our future work, we plan to extend this analysis with a more formal approach.

Our model had constant population of agents (although we assessed the impact of temporal failures on our distributed algorithm, see Section 8.7). However, it can be easily extended to realistic systems with new agents joining and existing agents permanently leaving the system. We proposed dynamic protocols for such a system in [Skowron and Rzadca 2013] and validated them using a prototype implementation.

Storage contracts should be signed for a pre-determined time period (e.g., one week). In the background, to replace the expiring contracts and to handle changes in partners' availabilities, agents should run the distributed matching algorithm (that will determine the "next" assignment). A contract would be replaced only if the gain in the availability balances the bandwidth needed to transfer the data. Such a system naturally responds to agents permanently leaving the system, as their availability would gradually diminish. Newcomers should enter the system with zero availability (otherwise, a Sybil attack would be possible). The newcomers will be adopted by highly-available agents: the newcomers will gradually replace agents that no longer need to be adopted; also, some adoption slots can be reserved, so that a newcomer is immediately assigned to a highly-available agent.

Some agents, instead of running our software on their computers, may choose to run it on resources rented from IaaS (Infrastructure as a Service) providers (see also experiments in Section 8.4). For such agents, our system can offer additional resilience to providers' permanent failures (resulting from, e.g., providers going bankrupt). At the same time, we think that only a minority of users would rent resources (because of monetary costs). Our system would enable this minority to interact with the large user-base relying on free services, and thus leverage the network effect.

9.1. Extension: Storage heterogeneity

To make our model mathematically tractable, we assumed that all agents have the same storage space, s. Under this assumption, in the subgame-perfect equilibrium agents replicate data of agents with similar availability (Proposition 5.2). In a realistic system, agents may dedicate to the system different amounts of storage—however, this heterogeneity does not significantly alter the resulting equilibrium (if the storage size is significantly smaller than the population size). We experimentally assessed heterogeneous storage in Section 8.5; here, we briefly present how to extend our analytical results.

Assume that each agent has s_i available replication slots ($1 \leq s_i < n$). The subgame-perfect equilibrium can be constructed by an algorithm that, first, orders agents by their availabilities u_i, then, for each agent i starting with the most available agent u_1, constructs s_i replication agreements with highest-available agents that are not yet "full" (have all their replication slots taken). The algorithm results in a subgame-perfect equilibrium by a similar argument as in Proposition 5.2. The most available agent p_1 has s_1 replication agreements with agents $p_2, \ldots, p_{s1+1}$ (if agent 1 replicated data of some agent $i > (s_1+1)$ instead of $j \leq (s_1+1)$, it would be optimal both for agent 1 and an agent j to change their replication agreements to form an agreement between 1 and j). For any other agent i, not following the algorithm, i.e., picking agent k instead of j, one of s_i currently most available agents, would result in a non-optimal allocation both for i and j and therefore, i and j would both have an incentive to deviate.

As a consequence, unlike as in the homogeneous storage model, agents will not form replication cliques. However, under a realistic assumption that the number of replication slots each agent has is small compared to the size of the population, agents still replicate with other agents having similar availability. Assuming that the population of agents is large, an agent with unavailability u_i will form s_i replication agreements with agents having similar unavailability. Thus, the resulting data unavailability is still $d_i = {u_i}^{s_i+1}$; which, similarly to the homogeneous model, makes the price of anarchy unbounded (Proposition 5.5 still holds).

The complexity of equitable optimization in the centralized model with heterogeneous storage size is still NP-hard (as the analyzed homogeneous model is a special case of the heterogeneous model).

The game-theoretic mechanism (Section 6.2) is still truthful in the heterogeneous model as long as the number of slots used by the mechanism r is at least equal to the minimal storage size: each agent has an incentive to increase the number of replication slots as it will result in additional replication contracts and thus increase its data availability. This result suggests that the system must be able to enforce a minimal storage size among the agents (similarly to enforcing that agents indeed donate r slots).

10. CONCLUSIONS

We studied the problem of replica placement in a decentralized storage system in order to optimize availability. We argued that replication should be based on cliques of agents replicating each others' data, rather than on a directory or bilateral assignments. We analyzed an idealistic model of agent availability that focuses on uncertainty of agent's on-line status. We proved that it is NP-hard to optimize availability for the *socially-equitable* scheme (in which the data availability of all agents is similar). We also analyzed a game theoretic version of the problem in which agents form bilateral replication agreements. We demonstrated that the loss in the global efficiency compared to the socially-optimal solution (the *price of anarchy*) is arbitrarily large.

In order to reduce the price of anarchy, we proposed a semi-centralized "adoption" mechanism: highly-available agents donate some of their replication slots. The algorithm uses these slots to replicate data of weakly-available agents. We formulated rules that guarantee that this algorithm is a truthful mechanism: no agent would obtain higher data availability by lowering its declared availability.

We proposed heuristics for centralized and decentralized matching that perform well in simulation experiments. In particular, the "adoption" heuristics increases the data availability of the weaker agents by two to three orders of magnitude, which should satisfy basic storage services.

Our results have practical consequences for decentralized storage or replication systems. Most importantly, in such systems, if allowed to choose partners by themselves, highly available agents will tend to replicate data among each other, and to exclude agents with low availability. This could result in unacceptable performance for agents with lower availabilities (in our experiments, less than about 30%). While this phenomenon provides an incentive for agents to be highly available, it can be also discouraging for the newcomers to join, and thus—hard for the system to gain momentum and large scale.

The system thus needs a method of availability redistribution — for instance, the proposed adoption algorithm. However, such methods have a problem analogous to any taxation and social security system. While we have proved that the mechanism is truthful, we did it on a model assuming that all agents would participate. Obviously, for highly-available agents, not donating slots for adoption (not paying taxes) dominates any adoption (paying taxes, or any redistribution mechanism). In real-world societies, this problem is solved by tax law enforcement; in decentralized storage, a combination of a reputation system and default values of parameters in the software must be sufficient (analogously to the Internet at large).

Currently, we are working on an implementation of a decentralized storage system with reciprocal storage contracts (nebulostore.org). The principal design goal is to provide a storage layer for a distributed on-line social networking software, such as PeerSoN [Buchegger et al. 2009].

REFERENCES

B. Amann, B. Elser, Y. Houri, and T. Fuhrmann. 2008. IgorFs: A Distributed P2P File System. In *P2P, Proc.* 77–78.

AmazonS3 2013. Amazon S3, Cloud computing storage for files, images, videos. http://aws.amazon.com/s3/. (2013).

SF Assmann, DS Johnson, DJ Kleitman, and J.Y.T. Leung. 1984. On a dual version of the one-dimensional bin packing problem. *Journal of algorithms* 5, 4 (1984), 502–525.

G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. 1999. *Complexity and approximation*. Springer.

M. Babaioff, J. Chuang, and M. Feldman. 2007. *Algorithmic Game Theory*. Cambridge, Chapter Incentives in Peer-to-Peer Systems, 593–611.

Samuel Bernard and Fabrice Le Fessant. 2009. Optimizing Peer-to-Peer Backup using Lifetime Estimations. In *Damap Proc.*

R. Bhagwan, S. Savage, and G. Voelker. 2002. Replication strategies for highly available peer-to-peer storage systems. *Fu-DiCo, Proc.* (2002).

R. Bhagwan, S. Savage, and G.M. Voelker. 2003. Understanding availability, In IPTPS, Proc. *LNCS* 2735 (2003), 256–267.

R. Bhagwan, K. Tati, Y.C. Cheng, S. Savage, and G.M. Voelker. 2004. Total recall: System support for automated availability management. In *NSDI, Proc.*

BitTorrentSync 2013. http://labs.bittorrent.com/experiments/sync.html. (2013).

Stevens Le Blond, Fabrice Le Fessant, and Erwan Le Merrer. 2012. Choosing partners based on availability in P2P networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7, 2 (2012), 25.

A. Bogomolnaia and M.O. Jackson. 2002. The stability of hedonic coalition structures. *Games and Economic Behavior* 38, 2 (2002), 201–230.

William J Bolosky, John R Douceur, David Ely, and Marvin Theimer. 2000. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 28. ACM, 34–43.

W. J. Bolosky, J. R. Douceur, and J. Howell. 2007. The Farsite project: a retrospective. *ACM SIGOPS Operating Systems Review* 41 (2007), 17–26.

S. Buchegger, D. Schiöberg, L.H. Vu, and A. Datta. 2009. PeerSoN: P2P social networking: early experiences and insights. In *ACM SNS, Proc.*

J. Busca, F. Picconi, and P. Sens. 2005. Pastis: A highly-scalable multi-user peer-to-peer file system. In *Europar, Proc.* 1173–1182.

F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems* 26 (2008), 4:1–4:26.

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

I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. 2001. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*. 46–66.

L.P. Cox, C.D. Murray, and B.D. Noble. 2002. Pastiche: Making backup cheap and easy. *ACM Operating Systems Rev.* 36 (2002), 285–298.

J.R. Douceur and R.P. Wattenhofer. 2001a. Competitive hill-climbing strategies for replica placement in a distributed file system, In DISC, Proc. *LNCS* 2180 (2001), 48–62.

John R Douceur and Roger P Wattenhofer. 2001b. Optimizing file availability in a secure serverless distributed file system. In *RDS, Proc.* IEEE, 4–13.

J.H. Drèze and J. Greenberg. 1980. Hedonic coalitions: Optimality and stability. *Econometrica: Journal of the Econometric Society* (1980), 987–1003.

A. Fabrikant, A. Luthra, E. Maneva, C.H. Papadimitriou, and S. Shenker. 2003. On a network creation game. In *PODC, Proc.* ACM, 347–351.

M. Feldman, K. Lai, J. Chuang, and I. Stoica. 2003. Quantifying disincentives in peer-to-peer networks. In *P2P Econ, Proc.*

F. Giroire, J. Monteiro, and S. Pérennes. 2009. P2P storage systems: How much locality can they tolerate?. In *LCN, Proc.* IEEE, 320–323.

Saikat Guha, Neil Daswani, and R. Jain. 2006. An experimental study of the skype peer-to-peer voip system. In *IPTPS, Proc.* 1–6.

S. Iyer, A. Rowstron, and P. Druschel. 2002. Squirrel: a decentralized peer-to-peer web cache. In *PODC, Proc.* 213–222.

M. Jelasity and O. Babaoglu. 2006. T-Man: Gossip-based overlay topology management, In ESOA, Proc. *Lecture Notes in Artificial Inteligence* 3910 (2006).

E. Koutsoupias and C. Papadimitriou. 1999. Worst-case equilibria, In STACS, Proc. *Computer Science Review* 1563 (1999), 404–413.

F. Le Fessant, C. Sengul, and A.M. Kermarrec. 2008. *Pace-maker: Tracking peer availability in large networks*. Technical Report RR-6594. INRIA.

Zhenhua Li, Jie Wu, Junfeng Xie, Tieying Zhang, Guihai Chen, and Yafei Dai. 2011. Stability-Optimal Grouping Strategy of Peer-to-Peer Systems. *Parallel and Distributed Systems, IEEE Transactions on* 22, 12 (2011), 2079–2087.

P. Linga, I. Gupta, and K. Birman. 2003. A Churn-resistant Peer-to-peer Web Caching System. In *SSRS, Proc.* 1–10.

Xin Liu and Anwitaman Datta. 2012. Contextual trust aided enhancement of data availability in peer-to-peer backup storage systems. *Journal of Network and Systems Management* 20, 2 (2012), 200–225.

C. Hsu M. Hefeeda and K. Mokhtarian. 2008. pCache: A Proxy Cache for Peer-to-Peer Traffic. In *SIGCOMM, Proc.*

P. Michiardi and L. Toka. 2009. Selfish Neighbor Selection in Peer-to-Peer Backup and Storage Applications. In *Euro-Par, Proc. (LNCS)*, Vol. 5704.

J.W. Mickens and B.D. Noble. 2006. Exploiting availability prediction in distributed systems. In *NSDI, Proc.*

T. Moscibroda, S. Schmid, and R. Wattenhofer. 2006. On the topologies formed by selfish peers. In *PODCS, Proc.* ACM, 133–142.

A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. 2002. Ivy: A Read/Write Peer-to-Peer File System. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 31–44.

M.J. Osborne. 2004. *An introduction to game theory*. Oxford University Press.

L. Pamies-Juarez, P. Garcia-Lopez, and M. Sanchez-Artigas. 2011. Enforcing fairness in P2P storage systems using asymmetric reciprocal exchanges. In *P2P, Proc.* IEEE, 122–131.

R. Rahman, T. Vinkó, D. Hales, J. Pouwelse, and H. Sips. 2011. Design space analysis for modeling incentives in distributed systems. In *ACM SIGCOMM, Proceedings*.

Shay Raz, Raz Lin, and Onn Shehory. 2008. Collaborative Load-Balancing in Storage Networks Using Agent Negotiation. In *Cooperative Information Agents XII*. Springer, 306–320.

R. Rodrigues and B. Liskov. 2005. High availability in DHTs: Erasure coding vs. replication, In IPTPS, Proc. *LNCS* 3640 (2005).

A. Rowstron and P. Druschel. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, In Middleware Proc. *LNCS* 2218 (2001), 329–350.

Krzysztof Rzadca, Anwitaman Datta, and Sonja Buchegger. 2010. Replica Placement in P2P Storage: Complexity and Game Theoretic Analyses. In *ICDCS, Proc.* IEEE Computer Society, 599–609.

SA 2013. Strategy Analytics: iCloud, Dropbox and Amazon top cloud media in the US. http://www.engadget.com/2013/03/21/strategy-analytics-cloud-media-market-share/. (2013).

Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. 2002. Taming aggressive replication in the Pangaea wide-area file system. *ACM SIGOPS Operating Systems Review* 36, SI (Dec. 2002), 15–30.

R. Sharma, A. Datta, M. DeH'Amico, and P. Michiardi. 2011. An empirical study of availability in friend-to-friend storage systems. In *P2P, Proc.* IEEE, 348–351.

W. Shi and Y. Mao. 2006. Performance evaluation of peer-to-peer Web caching systems. *Journal of Systems and Software* 79, 5 (2006), 714–726.

Piotr Skowron and Krzysztof Rzadca. 2013. Exploring heterogeneity of unreliable machines for p2p backup. In *HPCS 2013*.

R. G. Tinedo, M. S. Artigas, and P. G. Lpez. 2012. Analysis of data availability in F2F storage systems: When correlations matter.. In *P2P, Proc.* 225–236.

Laszlo Toka and Pietro Michiardi. 2011. Analysis of user-driven peer selection in peer-to-peer backup and-storage systems. *Telecommunication Systems* 47 (2011), 49–63. Issue 1.

K. D. Vohs, N. L. Mead, and M. R. Goode. 2006. The Psychological Consequences of Money. *Science* 314, 5802 (2006), 1154–1156.

Z. Yang, J. Tian, B.Y. Zhao, W. Chen, and Y. Dai. 2011. Protector: A Probabilistic Failure Detector for Cost-Effective Peer-to-Peer Storage. *TPDS* 22, 9 (2011), 1514–1527.

Z. Zhang, Q. Lian, S. Lin, W. Chen, Y. Chen, and C. Jin. 2007. BitVault: a highly reliable distributed data retention platform. *ACM SIGOPS Operating Systems Review* 41 (2007), 27–36.

Online Appendix to: Game-Theoretic Mechanisms to Increase Data Availability in Decentralized Storage Systems

Krzysztof Rzadca, Institute of Informatics, University of Warsaw, Poland
Anwitaman Datta, School of Computer Engineering, Nanyang Technological University, Singapore
Gunnar Kreitz, KTH Royal Institute of Technology, Sweden
Sonja Buchegger, KTH Royal Institute of Technology, Sweden

A. PROOFS OMITTED IN THE MAIN TEXT

PROPOSITION A.1. *The decision version of the Simple Stochastic Fair Replication Problem is NP-complete.*

PROOF. SSFRP is trivially in NP: given the grouping G_1 and G_2, it is sufficient to compute the corresponding products, which is in $O(n)$. The proof of hardness is by reduction from PARTITION [Ausiello et al. 1999]. In PARTITION, given a set of positive integers $\{b_i\}$, the goal is to decide whether it is possible to construct two disjoint subsets Y_1 and Y_2, such that $\sum_{b_i \in Y_1} b_i = \sum_{b_i \in Y_2} b_i = \frac{S}{2}$, where $S \triangleq \sum b_i$.

We construct an instance of SSFRP from an instance of PARTITION as follows. The i-th peer's unavailability corresponds to i-th integer $u_i = 2^{-b_i}$. Bound $B = 2 \cdot 2^{-S/2}$.

Given a solution Y_1, Y_2 to partition, the corresponding grouping $i \in G_1 \Leftrightarrow b_i \in Y_1$ is a solution to SSFRP. As $\sum_{b_i \in Y_1} b_i = \frac{S}{2}$, $\prod_{i \in G_1} 2^{-b_i} = 2^{-S/2}$. The same holds for G_2, thus $\prod_{i \in G_1} 2^{-b_i} + \prod_{i \in G_2} 2^{-b_i} = 2 \cdot 2^{-S/2} \leq B$.

Given a solution G_1, G_2 to SSFRP, the corresponding subsets are a solution to PARTITION. We denote as $2^{-l_1} = \prod_{i \in G1} 2^{-b_i}$ and $2^{-l_2} = \prod_{i \in G2} 2^{-b_i}$. Thus, $2^{-l_1} + 2^{-l_2} \leq 2 \cdot 2^{-S/2}$. We now show by contradiction that $l_1 = l_2 = S/2$. Assume $l_1 > l_2$. As $2^{-l_1} + 2^{-l_2} \leq 2^{1-S/2}$, $2^{-l_2} < 2^{1-S/2}$ (as $2^{-l_1} > 0$). Thus, $l_2 > S/2 - 1$ which leads to $l_2 \geq S/2$ (as l_2 is an integer). From the assumption of the proof, $l_1 > l_2$, thus $l_1 > S/2$, thus $l_1 + l_2 > S$, which leads to a contradiction, as by definition $S = \sum b_i = l_1 + l_2$. □

PROPOSITION A.2. *OSCA is NP-complete.*

PROOF. The proof is by reduction from DUAL BIN PACKING [Assmann et al. 1984]. In DUAL BIN PACKING, the task is to partition items with sizes $\{b_i\}$ into at least N disjoint sets (bins) $U_1, \ldots U_N$, so that the sum of elements in each set is at least B ($\sum_{b_i \in U_j} b_i \geq B$).

The reduction maps i-th integer b_i to i-th peer's unavailability $u_i = 2^{-b_i}$. The level B is mapped to maximum unavailability of the group $R = 2^{-B}$.

A solution $\{U_j\}$ of DUAL BIN PACKING is a valid solution of OSCA: $G_j = \{i \colon b_i \in U_j\}$, as $\sum_{b_i \in U_j} b_i \geq B$, thus $\prod_{i \in G_j} u_i = 2^{-\sum_{b_i \in U_j} b_i} \leq 2^{-B} \leq R$.

Similarly, a solution $\{G_j\}$ of OSCA is a valid solution of DUAL BIN PACKING ($B_j = \{b_i \colon i \in G_j\}$), as $\prod_{i \in G_j} u_i = 2^{-\sum_i b_i \in U_j b_i} \leq 2^{-B}$, thus $\sum_{b_i \in U_j} b_i \geq B$) □

B. ADDITIONAL EXPERIMENTS ON REAL TRACES

We performed additional experiments showing the performance of our algorithms on Microsoft PCs and Skype superpeer traces. Figures 10 and 11 show the results.

DOI: http://dx.doi.org/10.1145/0000000.0000000

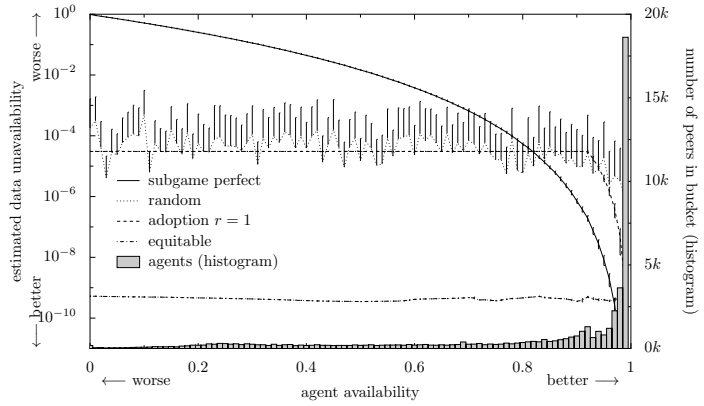

Fig. 10. Microsoft desktop PCs trace. Agents' expected data unavailability as a function of their availability in random, equitable, subgame perfect, and adoption assignment. Histogram shows the number of agents in each availability bucket.

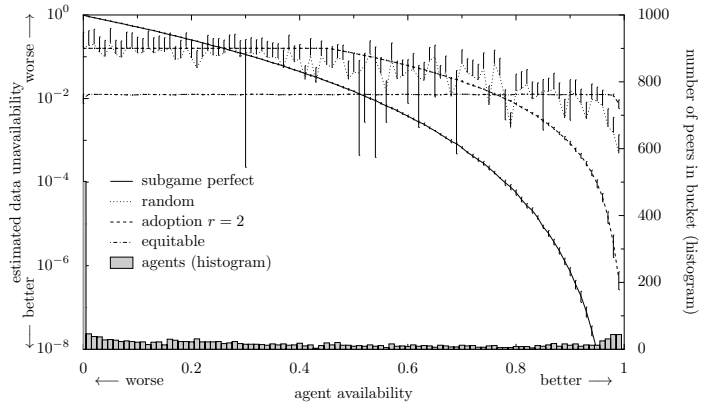

Fig. 11. Skype superpeers trace. Agents' expected data unavailability as a function of their availability in random, equitable, subgame perfect, and adoption assignment. Histogram shows the number of agents in each availability bucket.

The results confirm our results on stochastic traces: (i) in the subgame perfect equilibrium, a difference of data availability between weakly and highly available agents of a few orders of magnitude; (ii) the equitable allocation Pareto-dominates random allocation; (iii) the adoption mechanism able to guarantee acceptable availability for weakly-available peers.

These traces show two radically different distributions of availabilities: in the Skype trace, 25% of peers has less than 1% availability (and 14% — less than 0.1%) ; in contrast, in the Microsoft trace, 33% of machines are nearly always-on (availability greater than 99%).

Consequently, the adoption mechanism needs just $r = 1$ replication slot in the Microsoft trace to guarantee data unavailability $d_i \leq 0.0001$; whereas in the Skype trace, $r = 2$ replication slots guarantee just $d_i \approx 0.1$. The Skype results show that the adoption mechanism should perhaps enforce some minimal average availability required to join the system: in a month-long trace, 1% availability corresponds to just 7 hours of total on-line time.