DELIS
Dynamically Evolving, Large-scale Information Systems

Integrated Project
Member of the FET Proactive Initiative Complex Systems

DELIS-TR-627

Distributed Storage of Replicated Data

Olivier Beaumont, Philippe Duchon, Miroslaw Korzeniowski

2007
Distributed Storage of Replicated Data on Heterogeneous Platforms

Olivier Beaumont
LaBRI – INRIA Futurs
obeaumon@labri.fr

Philippe Duchon
LaBRI – Univ. Bordeaux
duchon@labri.fr

Miroslaw Korzeniowski
Wroclaw Univ. of Technology*
miroslaw.korzeniowski@pwr.wroc.pl

Abstract

We describe a fully decentralized algorithm, called “dating service” which produces uniformly chosen random matchings of linear size connecting resources of nodes in the system. Interestingly enough, this property holds true even if a node is not able to choose another node uniformly at random. In particular, the dating service can be implemented over existing DHT-based systems. In order to illustrate the expressiveness and the usefulness of proposed service, we propose an algorithm for distributed storage of replicated data that randomly distributes replicas over the network in logarithmic number of steps, that ensures that node storage capacities cannot be exceded.

1 Introduction

Peer-to-Peer overlay networks have proven their efficiency for storing and retrieving data. Structured Peer-to-Peer networks based on Distributed Hash Tables (DHT for short) such as Pastry [10] or Chord [12], provide efficient mechanisms for routing, but due to hashing, their use is limited to exact query searches. New services are therefore highly required in the context of large scale distributed networks. Moreover, heterogeneity is not taken into account in most systems and all resources play a symmetric role, whatever their capacity.

In this paper, we describe a distributed algorithm, called dating service. Its abstract purpose is to randomly join demands and supplies of some resource of many nodes into couples. In a round it produces a matching between demands and supplies which is of linear expected size (compared to optimal one) and is chosen uniformly at random from all matchings of this size.

We give a simple example of application. Namely, assigning in a random fashion data blocks owned by nodes to free memory cells offered by other nodes in order to create backup copies of data blocks.

The only result similar to ours of which we are aware is on choosing a random peer in a distributed hash table [6] by King and Saia. The authors give a method with which a peer can choose another one so that the distribution is uniform. The algorithm works in logarithmic time (expected and with high probability) and can also be run by many peers in a round yielding a linear number of connections distributed like balls thrown into $n$ bins. Since in such models, when the number of balls is also $n$, some bins get $\Omega(\log n/\log\log n)$ balls, in some situations our scheme has an advantage here, as it creates connections only if they are welcome on both sides. Another advantage is that our algorithm adapts easily to heterogeneous networks.

*The author is partially supported by Emerging Technologies Programme of the EU under EU Contract 001907 DELIS “Dynamically Evolving, Large Scale Information Systems” and by MNiSW grant number PBZ/MNiSW/07/2006/46. The work was done when the author was in LaBRI and INRIA Futurs in Bordeaux.
Schindelhauer and Shomaker [11], following the original idea of [5] have proposed two methods for weighted consistent hashing, that assign elements to nodes with probability proportional to their relative weight. The main advantage of the scheme we propose is that it ensures that storage capacities of nodes cannot be exceeded and that we consider the case of careful storage, where the replicas of a file are necessarily stored remotely, at the same logarithmic overall cost.

Several decentralized systems based on DHTs for distributed storage have been proposed in the literature, such as CFS [2], OceanStore [7], Ivy [8] and Glacier [3]. All these systems are DHT-based and provide reliability using replication, but they differ in the replication strategy. The indexes of nodes where data is replicated are either determined by some static strategies (replicating on the next \( r \) nodes) or using hashing. Thus, replicates may be stored on overloaded nodes and some extra mechanism is used to balance load afterwards, whereas the system we propose automatically matches replicated storage needs to storage availability.

### Notations

Throughout this paper we use the classical notations of \( O() \) and \( \Omega() \), where \( O(f(n)) \) means at most \( c \cdot f(n) \) for constant \( c > 0 \) and sufficiently large \( n \) and \( \Omega(f(n)) \) means at least \( c \cdot f(n) \) for constant \( c > 0 \) and sufficiently large \( n \).

Together with the above notations we also use the term with high probability (in short: whp) in the following way: a random variable \( X(n) \) is bounded by \( O(f(n)) \) with high probability means that for any constant \( \ell \), \( \Pr[X > c \cdot f(n)] \leq \frac{1}{n^\ell} \), where the constant \( c \) depends only on \( \ell \). The term for \( \Omega() \) notation is defined similarly.

### 2 Dating service

The dating service is a tool which provides some service to some applications. We describe it as a routine running completely independently of any applications, which means that we might have two separate networks, one for the dating service and one for an application and the latter can send requests to the former and receive answers.

The purpose of the dating service is to produce random matchings between two types of requests which, for convenience, we name supplies and demands. The routine works in rounds and in each round all peers of the application send information about their supplies and demands to the dating service, their requests are joined into couples and the peers can use these connections somehow and proceed to the next round.

In our description of the dating service, nodes taking part in providing the service are called servers, to distinguish them from their role as users. This is in no way an indication that peers are actually distinct from servers; in the end of this section, we describe how one can implement such a dating service in a distributed hash table.

The centralized dating service based on a single server would work as follows. All nodes of an application would submit information about all their supplies and demands to the server, the server would choose uniformly at random a maximum matching and send information about connections to all nodes of the application. Uniformly choosing a random matching in such a centralized way is actually trivial: just permute both types of requests and then match the \( i \)-th demand with the \( i \)-th supply for \( i \) between 1 and minimum of the number of supplies and demands.

In case when instead of a single server the dating service has \( n \) participating peers to its disposal, each peer works in exactly the same way and each piece of information about a supply or a demand is sent to a server chosen according to a fixed probability distribution \( P = (p_1, p_2, \ldots, p_n) \). A pseudocode of the dating service for a single round is shown in Algorithm 1.

Note that in the above description, it is not necessary that the random choice of servers be uniform – only that all requests are sent using the same distribution \( P \). This randomness is a load-balancing factor; as an extreme case, sending all requests to a single server would result in a centralized scheme.
Algorithm 1 Dating service (probability distribution $P = (p_1, ..., p_n)$)

for each peer $i$ in parallel do
  for each unit of supply or demand at $i$ do
    send a request to a random server chosen according to $P$
  for each server $j$ in parallel do
    $r \leftarrow$ number of demand requests at $j$
    $s \leftarrow$ number of supply requests at $j$
    $q \leftarrow \min\{r, s\}$
    choose uniformly at random without replacement $q$ requests of each type
    generate a uniformly chosen random perfect matching of the chosen requests
    for each of the $r + s$ received request do
      if the request is among the $2q$ chosen requests then
        send information about the partner to the creator
      else
        send to the creator information about no date

Denote the total number of demands by $m$ and the total supplies by $m'$. For the analysis, we assume that $m \leq m'$ (otherwise, just switch the symmetric roles of supply and demand); thus, $m$ is the maximum number of dates that a centralized service would be able to organize in a round.

We skip the proofs of the following lemmas. Some of them can be found in [1]. The first one states that the produced matching is symmetric, i.e. all demands are treated equally and so are all supplies.

Lemma 1. Conditioned on the total number of dates in a round of the dating service being $k$, the set of matches produced is a uniform random $k$-matching of the sets of demands and supplies.

The second lemma states that, provided the number of servers is not too large compared to the more numerous of supplies and demands, the dating service matches, on average, a constant fraction of the rarer type of requests with the other type. This result also holds with high probability provided that $m \in \Omega(\log n)$.

Lemma 2. Let $X = X(m, m', n, P)$ denote the number of dates organized by the dating service in a single round. Assume that $m' \geq m$ and $m' \geq cn$ for some positive constant $c$. Then there exists a constant $\beta = \beta(c) > 0$ such that, for any $P$, \[ E(X) \geq \beta m. \] (1)

Together, Lemmas 1 and 2 yield the following:

Lemma 3. Assume $m \leq m' = \Omega(n)$. Then, in any round of the dating service, each unit of supply has constant probability of being matched to a unit of demand.

Even though the events of different demands having dates in a single round are dependent, the events of a single demand getting a date over different, independent rounds are not. Thus, the following corollary easily follows.

Corollary 4. If the assumptions of Lemma 3 hold for each round, each single unit of supply gets matched with a demand within $O(\log n)$ rounds with high probability.

2.1 Dating Service in Peer-to-Peer Networks

In this subsection we describe how one can implement the dating service scheme in Peer-to-Peer networks. The idea is to use a distributed hash table (DHT; see for example [12, 9, 4, 10]). These schemes connect peers into networks and provide good routing strategies. The most important functionality for the dating service is the way that data is stored in such a network. In all designs there is an underlying virtual space (most often a $(0, 1]$ ring) which is partitioned among the peers. When a data item is inserted into a point in this virtual space, it is assigned to the peer responsible for this point.

One can easily adapt this to implement the dating service as follows. Whenever a request
must be sent to a $P$-random server, the sending peer simply picks a random key in the virtual space according to some common distribution (in the case of a $(0, 1]$ ring this is simply a uniform random variable on $(0, 1]$), and uses the DHT scheme to route the request to the responsible peer, which acts as a server; thus, the distribution of peer responsibilities in the virtual key space provides the common $P$ distribution. This distribution will typically not be uniform, but this is not assumed in our previous analysis.

To ensure that the dating service performs efficiently, the number of at least one type of requests (supply or demand) must be at least linear in the number of servers. To guarantee this condition, only peers with at least one request should act as servers in the dating service. With this precaution, we ensure that $m \geq n/2$ or $m' \geq n/2$, which is what is needed in our analysis.

Using a DHT has very important consequences for load balancing of the whole routine. Of course the dating service routine needs only small messages but it is still better to spread the work among many peers than to send everything to a single server. Even if one organises a DHT in the simplest way where nodes choose places on a $(0, 1]$ ring uniformly at random, the largest interval assigned to a peer will be of length $O((\log(n)/n)$ with high probability. This means that the expected fraction of requests received by a single peer (the maximum load of a peer) will exceed the average by an at most logarithmic factor. The dating service benefits also from such properties of DHTs as low congestion and routing time when routing even a linear number of messages to random places.

However, using a simple DHT still does not capture heterogeneity of the network. Strong peers participate in the DHT in the same manner as the weakest ones. One could adapt load balancing schemes based on virtual peers here but this is not the main topic of this paper. We emphasize however that if a DHT is designed which captures heterogeneity of peers, it should be easy to adapt it to the purposes of the dating service.

### 3 Application: Distributed Backup Storage

An example of an application of the dating service is the organization of a distributed backup scheme. Here each of $n$ peers in the network has some number of available unit size memory cells and some other number of unit size data blocks (large files being split into possibly many blocks), and we need to allocate some space for backup to each data block. Thus, demand is in the form of $m$ data blocks, and supply is in the form of a total of $m'$ memory cells. Assuming $m' \geq m$, we need to find a maximum (i.e., of size $m$) matching between them.

#### Careless backup

For the above defined problem the dating service works as follows. Call peers providers if they have some unused memory cells, and users if they have unassigned data blocks (a peer can be both a provider and a user). Each round, providers and users all take part in the dating service scheme (and act as servers) and submit their free memory cells and unassigned data blocks there. After each round, any peer which is no longer a user nor a provider stops participating in the dating service.

The following theorem is an application of Lemma 3, since the DHT shrinks together with decreasing number of providers and users.

**Theorem 5.** Assuming $m' \geq m$, our scheme based on the dating service matches each block with a memory cell in $O((\log(m)+\log(n))$ rounds, with high probability.

Notice that in the above description there is no termination detection mechanism. A simple way to implement it is to check how many providers and users there are (0 or many). If one of those types drops to zero, it means that the backup assignment is finished.

#### Careful backup

Our above solution to the distributed backup problem is very simple, but it also suffers from
a significant flaw: no effort is made to ensure that a peer is not assigned its own data blocks to backup, which makes no sense. Indeed, in the fully homogenous case \( n = m = m' \) (each peer having exactly one block to backup and one available memory cell), what our previous solution produces is a random permutation of blocks into cells, and it is well known that random permutations have at least one fixed point with probability close to \( e^{-1} \approx 0.366 \).

It is a simple matter to avoid such “self-backup” situation: any time the dating service produces a match between a block and cell originating from the same peer, the peer should simply disregard the match and submit its requests in future rounds (note that it is not needed to modify the dating service algorithm so that servers try to avoid such self-matches; while it would not be a difficult change, we would lose the uniformity property of Lemma 1).

With the above change, since our matching process is non-backtracking (i.e., once a block has been assigned to a memory cell, there are no provisions to undo this match), it is possible that the process will not terminate if demand too closely matches supply: it is conceivable that, at some point, only one user-provider remains. In such a situation, the process will go on indefinitely without creating any new matches.

It is not difficult to check that such a blocking situation is impossible if for each peer \( i \), the total demand (including that of peer \( i \)) is no more than the total supply excluding that provided by peer \( i \). If we denote the supply provided by peer \( i \) (its number of memory cells) by \( s_i \), the condition is simply

\[
m' \geq m + \max_{i} s_i. \tag{2}\]

Provided condition \( (2) \) is met, it is easy to prove that the process will terminate with probability 1, since, in each round where at least one unassigned block remains, at least one suitable unused cell remains and two such requests will be matched with some fixed, positive probability. This probability can be small, though, which means the system can enter an unfinished state that it will likely take a large number of rounds to leave.

Our next result shows that requiring slightly more than \( (2) \) ensures that the modified backup process also terminates in a logarithmic number of rounds, with high probability.

**Theorem 6.** With the above notations, assume

\[
m' \geq m + 2 \max_{i} s_i. \tag{3}\]

The modified backup process assigns each block to a suitable memory cell in \( O(\log(n) + \log(m)) \) rounds, with high probability.

### 4 Future Work

We strongly believe that the dating service can be used as a building block for many distributed services where nodes offer and request for resources. As an example we have given a distributed backup storage service. In [1] we have described how one can use the dating service for rumor spreading in heterogenous networks (our scheme achieves logarithmic time for all nodes and time \( \log_d n \) for nodes of bandwidths above average, if the average is \( d \)).

A possible extension consist to considering rumor mongering (or equivalently the broadcast of a large message). In this context, the message is split into smaller parts and is sent in a pipelined fashion through the network. In this case, we can make a deeper use of the dating service mechanism, since both incoming and outgoing bandwidths can be used efficiently. The most challenging problem consists in organizing the communications, so as to ensure that each part of the message is received exactly once.

We are preparing a result about generating a random graph in a distributed way. When each node declares its requested incoming and outgoing degree the dating service can uniformly choose a graph satisfying the degrees of one type and not exceeding the degrees of the other. In case when the total number of one type of requests exceeds the other type, the requests of the latter to be fulfilled are chosen uniformly at
random. Additionally we provide a scheme to mix the graph using the dating service, i.e. we change the graph slightly in each round, so that after logarithmic number of rounds it is almost independent of the original graph.

Last but not least the dating service works in rounds which is a usual assumption for first analyses but is unacceptable in a distributed system. We have started working on modelling the environment and algorithms so that they work in a partially asynchronous way, also in presence of node failures.

References


