Master’s Thesis

Replication in
Unstructured Peer-to-Peer Networks
with Availability Constraints

submitted by

Stefan Holder

on September 1, 2008

Supervisor
Prof. Dr. Gerhard Weikum

Advisor
Mauro Sozio (PhD)

Reviewers
Prof. Dr. Gerhard Weikum
Prof. Dr. Reinhard Wilhelm
Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, September 1, 2008

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, September 1, 2008
Acknowledgments

I would like to thank my advisor Mauro Sozio (PhD) for his persisting support and many fruitful discussions. Pursuing this thesis under his supervision gave me many insights into both theoretical and practical research. Moreover, I would like to thank Prof. Dr. Weikum for his valuable comments.
Abstract

Random walks have been proven a scalable search strategy for unstructured peer-to-peer (P2P) networks. In a random walk, a query message is forwarded at each step to a randomly chosen neighbor for a limited number of hops. If random walks are employed as a search strategy, data replication is an important method to increase the probability of successful search. In this context, we investigate the optimization problem to find a replica assignment to peers that satisfies given availability constraints and minimizes the number of replicas for each data item. Availability constraints are minimum thresholds for the probability of successful search and can be specified for each peer and each data item separately. In contrast to prior work, our approach aims to consider individual peer requirements regarding search success in the presence of temporary peer outages.

In this thesis, we present a distributed P2P replication algorithm for the above-mentioned optimization problem and prove an approximation guarantee for this algorithm. Our approach to devise such an algorithm is to first study the problem in a centralized setting (i.e. with complete knowledge of the network) and to develop a centralized algorithm. We then turn the centralized algorithm into a distributed one by exploiting local properties of the centralized algorithm. We show by simulations that the approximation solution is close to an optimum one in P2P settings. Another result of our evaluation is that the distributed algorithm efficiently performs replica assignments to peers on sparse network graphs with a large diameter. Mobile P2P networks, which are a prime example for such a graph topology, are therefore an interesting application area for our algorithm.
Contents

1 Introduction 1
   1.1 Search and Replication in P2P Networks . . . . . . . . . . . . . . . 1
   1.2 Motivation and Problem Statement . . . . . . . . . . . . . . . . . 4
   1.3 Approach and Contribution . . . . . . . . . . . . . . . . . . . . . 5
   1.4 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      1.4.1 Replication in P2P Networks . . . . . . . . . . . . . . . . 7
      1.4.2 Replication in Other Settings . . . . . . . . . . . . . . . . 9
   1.5 Outline of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Formal Model 12
   2.1 Model and Probabilistic Problem Statement . . . . . . . . . . . . . 12
   2.2 Deterministic Problem Statement . . . . . . . . . . . . . . . . . . 14

3 Centralized Algorithm 17
   3.1 Problem Transformation . . . . . . . . . . . . . . . . . . . . . . . 17
   3.2 Problem Complexity . . . . . . . . . . . . . . . . . . . . . . . . . 19
   3.3 Approximation Algorithm . . . . . . . . . . . . . . . . . . . . . . 20
   3.4 Approximation Guarantee . . . . . . . . . . . . . . . . . . . . . . 24
   3.5 Issues in P2P Networks . . . . . . . . . . . . . . . . . . . . . . . 29
   3.6 Experiments on the Approximation Quality . . . . . . . . . . . . . 30
      3.6.1 Simulation Setup and Implementation . . . . . . . . . . . . . 30
      3.6.2 Results and Discussion . . . . . . . . . . . . . . . . . . . . . 32
List of Figures

1.1 Importance of peer workloads and topology for replica placement . . . 6
3.1 Centralized algorithm in a P2P setting . . . . . . . . . . . . . . . . . . . 23
3.2 Approximation results for different edge probabilities . . . . . . . . . 33
3.3 Approximation results for up to 100 queries . . . . . . . . . . . . . . . 33
3.4 Approximation results for up to 8000 queries . . . . . . . . . . . . . . 34
3.5 Approximation results for different walk lengths . . . . . . . . . . . . . 34
3.6 Approximation results for different availability constraints . . . . . . . 35
3.7 Approximation results for different online probabilities . . . . . . . . . 35
4.1 The $l$- and $2l$-neighborhood of a peer for $l = 2$ . . . . . . . . . . . . 39
4.2 Low efficiency of the distributed algorithm . . . . . . . . . . . . . . . . 47
4.3 Efficiency results for different edge probabilities . . . . . . . . . . . . . 50
4.4 Graph diameter for different edge probabilities . . . . . . . . . . . . . 50
4.5 Efficiency results for different walk lengths . . . . . . . . . . . . . . . . 51
4.6 Efficiency results for up to 1000 queries . . . . . . . . . . . . . . . . . 52
4.7 Efficiency results for up to 20000 queries . . . . . . . . . . . . . . . . 52
4.8 Efficiency results for different availability constraints . . . . . . . . . . 52
4.9 Efficiency results for different online probabilities . . . . . . . . . . . . 53
List of Tables

2.1 Notation summary ................................................................. 13

3.1 Example of a matrix representing the sets $S_j$ .......................... 24
3.2 Default parameter values for evaluating the approximation quality . . 31
4.1 Default parameter values for evaluating the algorithm efficiency . . . 49
Chapter 1

Introduction

1.1 Search and Replication in P2P Networks

Peer-to-peer (P2P) computing is a “fundamental design principle for distributed systems” [SW05, p. 12], in which peers equally offer and request services. This is in contrast to the traditional client/server model, where services are only offered by servers and requested by clients. Moreover, P2P networks are characterized by decentralized self-organization and resource-usage [Ora01, SW05]. P2P applications cause more than 50% of internet traffic [Haß05] and include file sharing, IP-telephony, streaming media, and groupware [GKM+07].

P2P protocols are based on the notion of a P2P overlay network, which is a logical network on top of the physical network. In the overlay network, any two peers are connected by a logical communication link if there exists a path of physical communication links between both peers and both peers know each other. The peers, to which a peer is directly connected in the overlay network, are called the neighbors of this peer. Peer-to-peer networks are often classified according to the relationship between stored information and overlay topology [GKM+07], which leads to the distinction between structured and unstructured P2P networks.

In structured P2P networks, the location of data items is strongly coupled with the topology of the overlay network. Moreover, peers are connected in the overlay network in a way that enables the forwarding of queries towards the peer that stores the requested data. Protocols for structured P2P networks are mostly based on distributed hash tables (DHT). Examples for structured P2P networks using DHTs are Chord [SMK+01], Pastry [RD01], Tapestry [ZHS+04], and CAN [RFH+01]. DHTs enable finding data items within a bounded number of hops,
CHAPTER 1. INTRODUCTION

usually $O(\log(n))$, where $n$ is the number of peers in the network. However, structured P2P networks cannot easily cope with peers frequently joining or leaving the network, so-called churn. DHTs are primarily designed to support exact-match queries, although support for complex queries can be built on top of DHTs.

In unstructured P2P networks, peers can arbitrarily choose their neighbors and data items can be stored at any peer. New peers can therefore easily join the network, which makes the network resilient to high amounts of churn. The fact that peers can arbitrarily choose their neighbors has the advantage that peers can choose neighbors according to friendship or trust (e.g. in social networks) [MBSM05]. Moreover, it is easy to implement keyword queries and complex queries in unstructured P2P networks since peers use their local indexes for searching [RM06]. Chawathe et al. [CRB+03] argue that keyword queries are more common and important than exact-match queries, for instance in file sharing applications. On the other hand, since unstructured P2P networks lack an inherent routing mechanism, it is hard to find rare items in such networks using scalable search strategies. However, Chawathe et al. [CRB+03] argue that "most queries are for hay, not needles" [CRB+03, p. 408], meaning that most of the queries issued in P2P systems are for popular items.

One approach to implement search functionality in unstructured P2P networks is to use a centralized server for searching and indexing. A popular example of this architecture is Napster [nap], which was mainly used for file sharing of music (MP3) files. While searching is very easy to realize in this approach, the centralized architecture does not scale well and the centralized server is a single point of failure.

The (unstructured) Gnutella 0.4 protocol [gnu] does not specify any centralized component. Instead, bounded flooding is used for searching. In this search strategy, each peer forwards a search request to all of its neighbors for a limited number of hops, also called TTL (time-to-live). Bounding the flooding protocol is important since otherwise a huge amount of messages would be disseminated in the network, affecting considerably its performances. A disadvantage of this search strategy, which applies particularly to rare data items, is that bounded flooding does not guarantee finding the requested data item even if many peers are sampled. (A peer is sampled if it is traversed by a query message.) Moreover, flooding causes a high amount of duplicate messages, even in the case of duplicate detection [LCC+02].

In the semi-centralized approach, supernodes are used to overcome the scalability problems of flooding. Supernodes are peers with high bandwidth that store index information of other peers to reduce the number of required search messages. For instance, the Gnutella 0.6 protocol [KM02] and the FastTrack protocol, which is
used in KaZaA [kaz], employ supernodes. The semi-centralized approach has the disadvantage that some peers need to be capable and willing to act as supernodes. Moreover, a network with supernodes is more vulnerable to attacks than a fully decentralized network.

Another approach to implement scalable search in unstructured P2P networks is to employ random walks as a search strategy. In a random walk, the query message is forwarded at each step to a randomly chosen neighbor until the requested item is found or a maximum number of hops is reached. It has been shown both experimentally [LCC+02] and analytically [GMS04] that random walks give better performances than flooding. Since random walks are also bounded, there is the possibility that a query is unsuccessful even though the requested data item is stored on a peer.

The use of replication in unstructured P2P networks is appealing to research (e.g. [LCC+02, CS02, SNW08]) for a variety of reasons. In case bounded search strategies such as bounded flooding or random walks are employed, replication can be used to improve the probability of search success [SNW08] or to reduce the search traffic required to find data items [LCC+02, CS02]. Hence, to implement decentralized search without supernodes, replication is an adequate complement to bounded search strategies. Moreover, replication improves fault-tolerance since data is more likely to be available in the network in case of temporary or permanent node failure. Replication can also improve load balance since peers can choose from an increased number of replicas.

A widely recognized analysis of replication in unstructured P2P networks is the work of Cohen and Shenker [CS02]. They investigate the question what replication strategy minimizes the expected number of peers that have to be sampled until the requested data item is found. The main constraint in this setting is the storage capacities of peers. The result of this analysis is that an optimum solution is obtained if items are replicated in proportion to the square root of their global request rates. However, this approach does not explicitly address the question where to place replicas. Instead, it is assumed that replicas are uniformly distributed. Moreover, random sampling of nodes is used as an abstraction of concrete search protocols. While this is an elegant abstraction for the study of Cohen and Shenker, this approach is unsuitable to investigate where to place replicas for concrete search protocols.

Sozio et al. [SNW08] extend the approach of Cohen and Shenker [CS02] by explicitly addressing the question where to place replicas. To this end, they aim to maximize the overall probability of search success for given peer workloads, network topology, capacity constraints, and item sizes. The workload of a peer is the frequency of requests this peer has for each data item. Sozio et al. [SNW08]
CHAPTER 1. INTRODUCTION

4

propose a distributed replication algorithm that is tailored to realistic search protocols and achieves near-optimal replica placements with a proven performance guarantee. Moreover, the authors show with experiments that their approach is better suited to handle real-life peer workloads than the approach of Cohen and Shenker [CS02].

In this thesis, we also investigate where to place replicas in unstructured P2P networks with random walks as a search strategy. The research problem we study in this context is motivated and stated in the next section.

1.2 Motivation and Problem Statement

In this thesis, we investigate replication as a means to improve the probability of search success. A search success of 100% would be ideal and could be achieved by creating a replica of each item at each peer. However, this is usually not a viable option in practice since peers have limited storage capacities and this approach would cause significant network traffic. Thus, a natural question is what criteria should lead the decision how many replicas are needed and where should these be created.

Since peers in P2P systems are in general autonomous [Ora01], the design of P2P systems should be based on incentives for peers [SW05]. This means that if a peer participates in a P2P replication protocol and provides resources such as storage capacity or bandwidth, the peer expects a service in return. In our case, the service in return is an increased probability of search success. Moreover, peers reasonably expect the benefit they receive to be in proportion to the amount of resources such as storage capacity or bandwidth they provide.

It could be argued that approaches aiming to achieve a global optimum such as maximizing the overall probability of query success [SNW08] or minimizing the overall search traffic [CS02] provide an optimum solution for all peers together and hence each peer should be interested in achieving such a state. However, there is no guarantee for an individual peer that it will receive any benefit when joining such a network. For instance, in the approach of Sozio et al. [SNW08] where the overall probability of search success is maximized, some peers may receive no or very little benefit in terms of increased probability of search success. This may happen for peers that issue only few queries or that issue most of their queries for unpopular data items. From a global perspective, it is then too expensive to increase the probability of search success for these peers. Such a situation may appear to be unfair from the perspective of individual peers, particularly in the case when peers do not receive benefits although they provide a high portion of their resources.
In our approach, we allow the specification of minimum thresholds for the probability of search success. These probability thresholds are referred to as availability constraints and can be specified for each peer and each data item separately. Availability constraints are a means to achieve a fair balance between the amount of provided resources and received benefits for each peer. This can be achieved by setting the availability constraints of a peer in proportion to the amount of resources this peer provides.

Since availability constraints can be specified separately for each peer and each data item, availability constraints are also a means to take heterogeneous importance of individual peers and data items into account. This is particularly useful in scenarios where peers are less selfish and agree on performing a common task. Consider a P2P collaboration application used within a company’s intranet. In this scenario, peers or their respective users may have different priorities because of their job position (e.g. clerk or manager). Moreover, some items may be more important to peers/users than others and should therefore be more easy to find. For instance, an item that is related to the project on which a person is working is usually of higher importance than items of other projects. In all these cases, availability constraints can be assigned in proportion to the importance of users and data items in order to guarantee higher performances for users and data items of higher importance. Another application for availability constraints are commercial P2P systems, where availability constraints could be adjusted according to the price paid by users.

Our goal is to find a replica assignment to peers that satisfies given availability constraints and minimizes the number of replicas for each data item. Minimizing the number of replicas aims to reduce network traffic and is further motivated by the fact that peers only have limited storage capacities. Our objective is to devise a distributed algorithm that requires only local knowledge to achieve the desired replica assignment onto peers. In fact, the main result of our investigation is a distributed algorithm that is particularly suitable for sparse graphs with a large diameter. These characteristics make our algorithm particularly suitable for mobile P2P environments.

1.3 Approach and Contribution

Similar to Sozio et al. [SNW08], we consider individual peer workloads and the network topology to decide on the location of replica placements. This is because these parameters can significantly affect the minimum number of replicas required to satisfy given availability constraints. For instance, consider the network topology shown in Figure 1.1. In this network, random walks of length one are employed as a search strategy and only the three leftmost peers issue
queries for a particular data item $j$. All random walks (shown as arrows), on which the queries of the three query initiators are forwarded, traverse the white peer. Hence, by placing a replica of item $j$ at the white peer, we only need a single replica of item $j$ to satisfy all queries for $j$. Moreover, we observe that placing replicas of item $j$ on peers to the right of the white peer does not satisfy any query issued by the three query initiators. Intuitively, peers that likely to be sampled by many queries are good candidates for replica placement.

In addition to individual peer workloads and the network topology, we consider the online/offline behavior of peers. Peers can go offline because of failure, overload, or because the user switches off his or her computer. The online/offline behavior of peers makes a difference for replica placement since the time peers are online can vary considerably in P2P networks. Placing a replica at a peer that is likely to go offline shortly after the placement has the disadvantage that other replicas might then have to be created at other peers to maintain the availability constraints for the corresponding data item. First, this requires additional resources for further computations of replica placements and for actually performing the replica placements. Second, when a replica holder goes offline availability constraints for this data item can be violated for some time until new replica placements are computed and performed.

We investigate optimal replica placements for a steady state of the network. In a steady state, the network topology does not change. Moreover, the query patterns and the online/offline behavior of peers is stable with stochastic fluctuation but fixed (potentially unknown) parameters in the underlying stochastic processes. Using steady states for analysis is common in approaches that study replication in P2P unstructured networks such as [CS02, SNW08]. The dynamic behavior of real P2P networks can then be considered by re-computing the solution at appropriate time points (e.g. periodically).
We first study the problem where to place replicas in order to satisfy availability constraints in a centralized setting, that is, with complete knowledge of the network. We devise a probabilistic model in which we consider the probabilities of peers to be online and the probabilities of peers to issue queries for individual data items. Based on this probabilistic model, we formally state our problem. We then transform the probabilistic problem into a deterministic one. This deterministic problem can be formulated as a generalization of the partial set cover problem [Kea90, Pet94], which in turn is a generalization of the NP-hard set cover problem [Kar72, Joh73, Lov75, Chv79]. To the best of our knowledge, our generalization of the partial set cover problem has not been studied before. Because this problem is NP-hard, we devise a greedy approximation algorithm that is based on the intuition that the more likely it is for a peer to be sampled by queries, the more suitable this peer is for a replica placement. This approximation algorithm might also be useful for solving other problems beyond the scope of P2P networks. We prove an approximation guarantee for this algorithm and experimentally evaluate the quality of approximation solutions for different P2P settings.

We then turn the centralized algorithm into a distributed one by exploiting local properties of the centralized algorithm. We present an algorithm, which only requires local knowledge of the network and which is particular suitable for sparse graphs with a large diameter. We show that the centralized and the distributed algorithm are equivalent, meaning that for every input, the distributed algorithm produces the same replica assignment to peers as the centralized algorithm. Moreover, we empirically evaluate the efficiency of the distributed algorithm in terms of the time required to construct replica allocations.

Note that although we focus on single random walks as a search strategy, our results are also applicable to other search strategies such as multiple random walks per query [LCC+02] or bounded flooding. Although we focus on the physical replication of data items, our work is also applicable in the case of placing pointers that contain the address of the peer on which the item is actually stored.

## 1.4 Related Work

### 1.4.1 Replication in P2P Networks

The approach of Sozio et al. [SNW08] is closely related to our approach in terms of methodology. We use the same problem transformation to turn a probabilistic replication problem into a deterministic one. However, while the deterministic problem obtained by Sozio et al. [SNW08] is related to the multi-knapsack prob-
lem [CK05], we obtain a generalized version of the partial set cover problem [Kea90, Pet94].

The amount of replicated data needed to satisfy given availability constraints can be reduced using erasure codes [WK02] instead of replicating files completely. In this approach, a file is divided into \( m \) fragments and recoded into \( n \) fragments with \( m < n \). The file can then be reassembled from any \( m \) of the recoded fragments. However, erasure coding implies additional costs since the complete file needs to be reassembled from file fragments stored at multiple peers [LCL04].

Cuenca-Acuna et al. [CAMN03] use a special form of erasure coding, which makes it unnecessary to keep track of the fragments that have already been generated and placed at peers. In this approach, fragments can arbitrarily be generated and distributed since the probability that the same fragment is generated twice is very small. For each data item, fragments are generated and distributed until the desired overall availability has been reached. Compared to our approach, the current availability is computed using a simple availability model, which does not consider the network topology. This approach is based on the PlanetP platform [CAPMN03], where each peer has a membership directory that contains names and addresses of all peers in the network as well as a global content index containing term-to-peer mappings. While this approach enables peers to perform searching locally, it clearly has scalability limitations since each peer needs to have global knowledge of the network.

Replication is also applied in structured P2P networks. For instance, Beehive [RS04] and EpiChord [LLD06] are approaches tailored for DHT routing mechanisms.

DHTs usually rely on consistent hashing [KLL+97]. This hashing scheme has the advantage that in case peers join or leave the network, only little effort is required for maintaining the DHT. Brinkmann et al. [BEMS07] show that consistent hashing is not perfectly suited for replication if members have heterogeneous storage capacities and the replication strategy is required to be fair, that is, a member with \( x\% \) of the available storage capacity gets \( x\% \) of the data. Brinkmann et al. therefore propose a data placement strategy, coined Redundant Share, that achieves fairness in the presence of heterogeneous storage capacities. However, this placement strategy is targeted for storage area networks and assumes that each member has complete knowledge of the network. Mense and Scheideler [MS08] propose an alternative data distribution scheme called SPREAD for fair data replication and heterogeneous storage capacities with an improved adaptivity. However, their approach requires also complete knowledge of the network. Brinkmann and Effert [BE08] extend the Redundant Share algorithm of Brinkmann et al. [BEMS07] for P2P environments, where each peer knows a
limited number of other peers, called its view. Under the assumption that each peer knows the remaining storage capacities of all peers in its view, this approach enables each peer to locate all replicas inside its view. In contrast to our work, Brinkmann and Effert [BE08] do not explicitly consider replication to improve the availability of data items in P2P networks.

Finally, Morselli et al. [MBSM05] propose a search and replication protocol for unstructured P2P networks that is inspired by the routing mechanism of DHTs and also addresses the question where to place replicas. In this approach, each query is sent on a random walk followed by a “deterministic” walk. In the deterministic walk, the query message is forwarded until a so-called “local minimum” for the requested data item is reached. Hence, replicas only need to be placed at their local minima. A disadvantage of this approach is that the number of local minima may be large. Peers might then not have sufficient storage capacities to store all data items for which they are a local minimum [SNW08]. Sozio et al. [SNW08] show experimentally that their approach achieves higher availability of data items, especially in the case of real-life peer workloads.

1.4.2 Replication in Other Settings

The problem of replica placement has also been studied in the context of distributed database systems, where transactional data is replicated (e.g. [Rie98, JR94]). In these approaches, a probabilistic graph is considered with failure-prone nodes and/or failure-prone communication links and the availability of data items is examined. In this case, search strategies are not explicitly taken into account since the location of data items is assumed to be known. In contrast to our approach, it does therefore not matter how far away nodes that store the requested data are as long as there exists a path of functioning communication links and nodes between the requesting node and a resource holder.

The problem of optimal resource placement in a graph with error-prone communication links is proven to be \#P-complete [JR94], that is, to be at least as hard as NP-complete problems. Optimal resource placement in this context means placing a single resource such that the availability is maximized, given global request rates of data items.

In the approach of Riedewald [Rie98], nodes are assumed to be subject to failure and communication links are assumed to not fail as in our approach. (However, a communication link can only function if both nodes it connects also function.) The objective is to minimize the number of replicas needed to achieve given global availability constraints for data items, which is similar to our approach. Moreover, nodes are also assumed to have unlimited storage capacities as in our approach.
The main result of this work is a heuristic algorithm for replica placement, which is similar to our centralized greedy algorithm and therefore summarized below.

The algorithm computes the replica assignment to peers separately for each data item. Let \( a_j(v) \) be the probability that data item \( j \) is accessible from peer \( v \) in the network and let \( a_j \) be the probability that data item \( j \) is accessible given the request probability of each peer for this data item. It is assumed that the network graph is acyclic because with this assumption \( a_j(v) \) can be computed for each peer with a simple deep-first search algorithm. Then \( a_j \) is computed as the weighted sum of the \( a_j(v) \) values of all peers with weights according to the request probabilities of all peers for data item \( j \). When the algorithm starts, it is assumed that no replica has been placed yet. At each step, one more replica is placed by first computing \( a_j \) for each candidate placement of the new replica. Then the candidate placement with the highest increase on \( a_j \) is chosen. Finally, the algorithm stops as soon as the desired availability constraint for item \( j \) is satisfied.

Riedewald shows with experiments that the proposed heuristic works well for various network topologies but no approximation guarantee is given. Compared to our approach, we do not require graphs to be acyclic. Moreover, Riedewald’s heuristic computes the replica placements for each data item in a centralized way, similar to our centralized algorithm. To parallelize the algorithm, Riedewald suggests to compute the replica placements of different data items on different nodes. In contrast, our distributed algorithm computes the replica placements for each data item in a distributed way.

In [GWWK00, GWW02], the automatic configuration of workflow management systems is studied. In particular, the authors address the question how often servers need to be replicated in order to meet user-specified guarantees for the quality of services in terms of performance, availability, and performability (a mixture of performance and availability). The objective is to achieve the specified quality guarantees with minimum total cost for replicating servers. In this setting, the network is not considered as each server is assumed to be accessible whenever it is functioning. Regardless of this different scenario, the replication problem is similar to that of Riedewald [Rie98]. Moreover, a greedy heuristic very similar to the one used by Riedewald is employed.

1.5 Outline of the Thesis

The remainder of this thesis is structured as follows. In Chapter 2, we present a formal model and a formal problem statement. In Chapter 3, we devise a centralized approximation algorithm for replication in unstructured P2P networks,
prove an approximation guarantee for this algorithm, and evaluate the approximation quality of this algorithm for P2P settings. In Chapter 4, we turn the centralized algorithm into a distributed one and evaluate the efficiency of the distributed algorithm in terms of the time needed to construct replica allocations. Finally, conclusions and future work are presented in Chapter 5.
Chapter 2

Formal Model

In this chapter, we define a formal model and formally state our problem. In Section 2.1, the model is presented and a probabilistic problem is formulated. In Section 2.2, the probabilistic problem is turned into a deterministic one.

2.1 Model and Probabilistic Problem Statement

The overlay network topology is modeled as an undirected graph $G = (V, E)$ with $n$ vertices such that each peer corresponds to a vertex. Peers can only communicate if their corresponding vertices are neighbors in this graph. Moreover, there are $k$ distinct data items in the network. A replica allocation $A_j \subseteq V$ for a data item $j$ is the set of peers that store a replica of item $j$. Hence, each peer stores at most one replica of each item. Note that we do not take storage capacities of peers explicitly into account. However, we consider storage capacities by the fact that we aim to minimize the number of replicas for each data item, which is explained later in this section in more detail. Otherwise, choosing $A_j = V$ for all $j = 1, \ldots, k$ would trivially satisfy all availability constraints. See Table 2.1 for a summary and reference of the notation used in this chapter.

In our model, time is running in discrete steps. In each of these time steps, each peer $v_i$ is online with probability $o_i$, independently from other peers. We refer to $o_i$ as the online probability of peer $v_i$. When a peer goes online again after any number of temporary outages, it chooses the same neighbors as before.
Table 2.1: Notation summary

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (V, E)$</td>
<td>Network topology</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of peers</td>
</tr>
<tr>
<td>$v_i$</td>
<td>Peer</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of data items</td>
</tr>
<tr>
<td>$j$</td>
<td>Data item</td>
</tr>
<tr>
<td>$A_j$</td>
<td>Replica allocation for item $j$</td>
</tr>
<tr>
<td>$o_i$</td>
<td>Online probability of peer $v_i$</td>
</tr>
<tr>
<td>$q_{ij}$</td>
<td>Query rate of peer $v_i$ for item $j$</td>
</tr>
<tr>
<td>$l$</td>
<td>Random walk length</td>
</tr>
<tr>
<td>$W$</td>
<td>Random walk</td>
</tr>
<tr>
<td>$p_{ij}$</td>
<td>Availability constraint of peer $v_i$ for item $j$</td>
</tr>
<tr>
<td>$r_{Wj}$</td>
<td>Probability that a query for item $j$ goes through $W$</td>
</tr>
<tr>
<td>$Q$</td>
<td>Set of query executions</td>
</tr>
<tr>
<td>$Q_j$</td>
<td>Set of query executions for item $j$</td>
</tr>
<tr>
<td>$Q_{ij}$</td>
<td>Set of query executions initiated by peer $v_i$ for item $j$</td>
</tr>
<tr>
<td>$R_{ij}$</td>
<td>Set of query executions for item $j$ sampling peer $v_i$</td>
</tr>
</tbody>
</table>

Let $q_{ij}$ be the (time-invariant) probability that in any time step a query is issued by a peer $v_i$ for an item $j$, given that $v_i$ is online. More formally,

$$q_{ij} = \Pr[v_i \text{ issues a query for } j \text{ at any time step}| v_i \text{ is online}].$$

We refer to $q_{ij}$ as the query rate of peer $v_i$ for item $j$. In our setting, a query is also issued if a peer demands a data item that is locally available. However, the query is satisfied locally in this case.

A walk of length $l$ is a sequence of not necessarily distinct vertices $v_0, v_1, \ldots, v_l$ such that $(v_i, v_{i+1}) \in E$ for all $i = 0, \ldots, l - 1$. A random walk of length $l$ is a walk, starting in $v_0$, where at each step $i = 1, \ldots, l$ the vertex $v_i$ is chosen independently at random among the neighbors of $v_{i-1}$ that are currently online.

For each query that is issued by a peer, we have a query execution. A query execution is defined formally as a triple $(W, j, id)$, where $W$ is the random walk on which the query message is forwarded, $j$ is the requested item, and id is a unique id associated with the query. Hence, there may be many query executions for the same item with the same random walk. If the random walk $W$ of a query execution traverses a peer $v_i$, we say that the query execution samples peer $v_i$. Moreover, we say that a query execution $(W, j, id)$ is successful if the random
walk \( W \) traverses at least one peer that stores item \( j \). Otherwise, the query execution is not successful.

Searching is modeled by the following stochastic process. At each step, each peer \( v_i \) is online with probability \( o_i \). Moreover, at each step each peer \( v_i \) that is online issues a query for an item \( j \) with probability \( q_{ij} \). It is possible that no query is issued by any peer during a step, which is always the case if \( q_{ij} = 0 \) for each \( v_i \in V \) and for each \( j = 1, \ldots, k \). Moreover, the number of queries issued at each step is at most \( nk \). This upper bound is reached in every step if \( q_{ij} = 1 \) for each \( v_i \in V \) and for each \( j = 1, \ldots, k \). If a peer \( v_i \) issues a query for an item \( j \), we have a query execution initiated by peer \( v_i \) for item \( j \) with a random walk that has a maximum length of \( l \). The random walk on which the query message is forwarded ends as soon as it traverses a peer that stores item \( j \). This implies that in case peer \( v_i \) stores item \( j \), no query is sent through the network. Moreover, while the query is executed (i.e. the query message is forwarded), peers do not change their online states. Query executions are hence assumed to be very fast.

Finally, for each random walk \( W \) of length \( l \) and each item \( j \), denote by \( r_{Wj} \) the probability that a query issued for item \( j \) by the stochastic process goes through the random walk \( W \) during its execution.

We denote by \( p_{ij} \) the required minimum probability that a query issued by a peer \( v_i \) for an item \( j \) is successful in the stochastic process. The minimum probability \( p_{ij} \) is referred to as the availability constraint of peer \( v_i \) for item \( j \). A replica allocation \( A_j \) is said to satisfy the availability constraints for item \( j \) if the availability constraint \( p_{ij} \) is satisfied for each peer \( v_i \in V \).

With these ingredients, we can state our problem formally:

**Probabilistic replication problem (ProbabilisticReplication):** Given a P2P overlay network \( G = (V, E) \), online probabilities \( o_i \), query rates \( q_{ij} \), a set of data items, maximum walk length \( l \), and availability constraints \( p_{ij} \). The goal is for each item \( j \) to find a replica allocation \( A_j \) that (i) satisfies the availability constraints for item \( j \) and (ii) minimizes the number of replicas \( |A_j| \).

Since there are no capacity constraints in our model, the above optimization problem can be solved for each data item independently. Moreover, when minimizing the number of replicas for each data item, the total number of replicas is also minimized.

### 2.2 Deterministic Problem Statement

Let \( Q \) be a set of query executions in a network \( G = (V, E) \). We require that no data items are stored at the peers in \( V \). This is because a random walk with
maximum length \( l \) ends as soon as it traverses a peer that stores the requested data item. However, we aim to find optimal replica allocations regardless of the items that are currently stored in the network. (Note that we will remove this assumption in Section 4.1.) Let \( Q_j \subseteq Q \) be the set of query executions in \( Q \) for item \( j \). We have \( \bigcup_{j=1}^{k} Q_j = Q \). For each peer \( v_i \) and each item \( j \), let \( Q_{ij} \subseteq Q_j \) be the set of query executions initiated by peer \( v_i \) for item \( j \). Let \( R_{ij} \subseteq Q_j \) be the set of query executions for item \( j \) that sample peer \( v_i \). We have \( \bigcup_{i=1}^{n} Q_{ij} = \bigcup_{i=1}^{n} R_{ij} = Q_j \) for each item \( j = 1, \ldots, k \).

For each peer \( v_i \) and each item \( j \), we have an availability constraint \( p_{ij} \) with \( 0 \leq p_{ij} \leq 1 \). The availability constraint \( p_{ij} \) of a peer \( v_i \) for an item \( j \) is satisfied if the quotient between the number of successful query executions initiated by peer \( v_i \) for item \( j \) and the number of all query executions \( |Q_{ij}| \) initiated by peer \( v_i \) for item \( j \) is at least \( p_{ij} \). We have that \( Q_{ij} \cap \bigcup_{v_i' \in A_j} R_{i'j} \) is the set of query executions by peer \( v_i \) for item \( j \) that are satisfied since each of these query executions samples at least one peer \( v_i' \) holding a replica of \( j \). Therefore, the availability constraint \( p_{ij} \) of a peer \( v_i \) for an item \( j \) is satisfied if and only if

\[
|Q_{ij} \cap \bigcup_{v_i' \in A_j} R_{i'j}| \geq p_{ij} |Q_{ij}|
\]

A replica allocation \( A_j \) is said to satisfy the availability constraints for item \( j \) if the availability constraint \( p_{ij} \) is satisfied for each peer \( v_i \in V \), that is,

\[
|Q_{ij} \cap \bigcup_{v_i' \in A_j} R_{i'j}| \geq p_{ij} |Q_{ij}| \quad \text{for all } i = 1, \ldots, n.
\]

We can now formulate the deterministic replication problem:

**Deterministic replication problem (DeterministicReplication):** Given a network topology \( G \), a set of data items, a set of query executions \( Q \) performed in this network, and availability constraints \( p_{ij} \). The goal is for each item \( j \) to find a replica allocation \( A_j \) that (i) satisfies the availability constraints for item \( j \) and (ii) minimizes the number of replicas \( |A_j| \).

We do not have query rates and online probabilities in the DeterministicReplication problem because only a set of query executions is needed to find an optimal replica allocation for each data item. Since there are no capacity constraints, the deterministic optimization problem can be solved by solving \( k \) optimization problems independently, one for each data item. Moreover, the total number of replicas is minimized when the number of replicas for each data item is minimized.

Any instance of the ProbabilisticReplication problem can be formulated as an instance of the DeterministicReplication problem by choosing a set of query executions \( Q \) as follows. (Note that in the DeterministicReplication problem, we
have an arbitrary set of query executions \( Q \). Let \( Q_{Wj} \subseteq Q_j \) be the set of query executions for item \( j \) with the random walk \( W \). We choose a set of query executions \( Q \) such that

\[
\frac{|Q_{Wj}|}{|Q|} = r_{Wj}
\]

for each \( j = 1, \ldots, k \) and each random walk \( W \) of length \( l \). In other words, the relative frequency of query executions in \( Q \) for an item \( j \) with a walk \( W \) equals its corresponding probability in our stochastic process for each item and each walk of length \( l \). Note that the same method for choosing \( Q \) could also be applied to other, possibly more sophisticated, probabilistic models of searching in P2P networks.

If \( Q \) gives a good approximation of an instance of the ProbabilisticReplication problem then an optimum solution to the corresponding deterministic replication problem is also an optimum solution to the probabilistic one. In the remainder of this thesis, we exploit this property and focus on solving the DeterministicReplication problem.

In Chapter 3, we develop a centralized algorithm for the DeterministicReplication problem and in Chapter 4, we turn this centralized algorithm into a distributed one. In Chapter 4, we also present a method for obtaining a set of query executions \( Q \), the input of our algorithm, in P2P networks.
Chapter 3

Centralized Algorithm

In this chapter, we investigate the deterministic replication problem (DeterministicReplication) in a centralized setting, that is, with complete knowledge of the network. We first show that the DeterministicReplication problem can be formulated as a generalization of the NP-hard partial set cover problem. Then we propose a centralized approximation algorithm and prove an approximation guarantee for this algorithm. Finally, we experimentally evaluate the approximation quality of our algorithm for different P2P settings.

3.1 Problem Transformation

In this section, we show that the DeterministicReplication problem can be formulated as a generalization of the NP-hard partial set cover problem [Kea90, Pet94].

Let \( U = \{U_1, U_2, \ldots, U_{n'}\} \) be a family of finite and pairwise disjoint ground sets. Let \( S \) be a family of subsets of \( \bigcup_{t=1}^{n'} U_t \), such that \( S \) is a cover of \( \bigcup U_t \), i.e. \( \bigcup_{S \in S} S = \bigcup U_t \). Let \( \varphi = \{p_1, p_2, \ldots, p_{n'}\} \) with \( 0 \leq p_t \leq 1 \) for all \( t = 1, \ldots, n' \).

We say that a subset \( S^* \subseteq S \) is a \( \varphi \)-partial cover of \( U \) if in each ground set \( U_t \in U \) the sets in \( S^* \) cover at least \( \lceil p_t |U_t| \rceil \) elements, that is,

\[
|U_t \cap \bigcup_{S \in S^*} S| \geq p_t |U_t| \quad \text{for all } t = 1, \ldots, n'.
\]

Partial set cover problem with multiple ground sets (MultiPartialSetCover): Given a family of finite and pairwise disjoint ground sets \( U = \{U_1, U_2, \ldots, U_{n'}\} \), a finite cover \( S \) of \( \bigcup U_t \), and \( \varphi = \{p_1, p_2, \ldots, p_{n'}\} \),
with \(0 \leq p_t \leq 1\) for all \(t = 1, \ldots, n'\), the goal is to find a minimum-size subset \(S^* \subseteq S\) that is a \(\varphi\)-partial cover of \(U\).

The MultiPartialSetCover problem is a generalization of the partial set cover problem \([\text{Kea90}, \text{Pet94}]\), which in turn is a generalization of the set cover problem \([\text{Kar72}, \text{Joh73}, \text{Lov75}, \text{Chv79}]\). The partial set cover problem and the set cover problem are defined as follows:

**Partial set cover problem (PartialSetCover):** Given a finite ground set \(U\), a finite family \(S\) of subsets of \(U\) such that \(S\) is a cover of \(U\) (i.e. \(\bigcup_{S \in S} S = U\)), and \(0 \leq p \leq 1\). The goal is to find a minimum-size subset \(S^* \subseteq S\) that covers at least \(\lceil p|U| \rceil\) elements of \(U\), that is,

\[
\left| \bigcup_{S \in S^*} S \right| \geq p|U|.
\]

**Set cover problem (SetCover):** Given a finite ground set \(U\) and a finite family \(S\) of subsets of \(U\) such that \(S\) is a cover of \(U\) (i.e. \(\bigcup_{S \in S} S = U\)), the goal is to find a minimum-size subset \(S^* \subseteq S\) that is a cover of \(U\).

The MultiPartialSetCover problem contains the PartialSetCover problem as a special case for \(n' = 1\). The generalization of the PartialSetCover problem to the MultiPartialSetCover problem can be regarded as partitioning the ground set of the PartialSetCover problem into multiple disjoint ground sets and specifying a partial cover constraint for each ground set partition separately. Moreover, the PartialSetCover contains the SetCover problem as a special case for \(p = 1\).

Recall that the DeterministicReplication problem can be solved by solving \(k\) optimization problems independently, one for each data item (see Section 2.2). For a single item \(j\), we can now formulate the DeterministicReplication problem in terms of the MultiPartialSetCover problem. This shows that we can transform each instance of the DeterministicReplication problem into \(k\) instances of the MultiPartialSetCover problem.

Let \(j\) be the item for which we formulate the DeterministicReplication problem in terms of the MultiPartialSetCover problem. A query execution for item \(j\) in the DeterministicReplication problem is an element in the MultiPartialSetCover problem, i.e. \(Q_j = \bigcup U_i\). Each set of query executions \(Q_{ij}\) initiated by a peer \(v_i\) for item \(j\) is a ground set \(U_i \in U\). Hence, \(\{Q_{1j}, \ldots, Q_{nj}\} = \{U_1, \ldots, U_{n'}\} = U\), which implies \(n = n'\). Moreover, each set of query executions \(R_{ij}\) for item \(j\) that sample a peer \(v_i\) is a set \(S \in S\), i.e. \(\{R_{1j}, \ldots, R_{nj}\} = \{S_1, \ldots, S_n\} = S\). The availability constraint \(p_{ij}\) for a peer \(v_i\) and item \(j\) is \(p_i\) in the MultiPartialSetCover problem for all \(i = 1, \ldots, n\). A replica allocation \(A_j \subseteq V\) then corresponds to a subset \(S^* \subseteq S\); hence, minimizing the number of replicas \(|A_j|\) is equivalent
to minimizing the size of the subset $S^*$. Moreover, satisfying the availability constraints for item $j$ is equivalent to having a $\varphi$-partial cover of $U$.

### 3.2 Problem Complexity

The SetCover problem and the PartialSetCover problem are both NP-hard optimization problems. Since the MultiPartialSetCover problem is a generalization of the PartialSetCover problem (and hence also a generalization of the SetCover problem), it is also NP-hard.

Because the DeterministicReplication is very similar to the MultiPartialSetCover problem (see previous section), we conjecture the following:

**Conjecture 1.** The optimization problem DeterministicReplication is NP-hard.

A common method to prove the NP-hardness of a problem is to reduce another problem that is already known to be NP-hard to the first problem with a polynomial-time many-one reduction (see Papadimitriou [Pap94], for instance). Because the DeterministicReplication problem is very similar to the MultiPartialSetCover problem, the MultiPartialSetCover problem seems to be a good candidate for such an NP-hard problem. However, it is not straightforward to transform each instance of the MultiPartialSetCover problem into an instance of the DeterministicReplication problem since the set of query executions $Q$ is subject to constraints such as the graph topology and the fact that walks have a fixed length. Another constraint of $Q$ is that each query execution samples the peer that issued the query. Hence, in the DeterministicReplication problem, we have $U_{ij} \subseteq S_{ij}$ for each $v_i \in V$ and each $j = 1, \ldots, k$. A better candidate to show the NP-hardness of the DeterministicReplication problem might be the SetCover problem, because this problem is a special case of the MultiPartialSetCover problem (see previous section) and it has also obvious relations to the DeterministicReplication problem.

Since polynomial-time algorithms are unlikely to exist for NP-hard problems, our goal is to devise a polynomial-time approximation algorithm for the MultiPartialSetCover problem. This approximation algorithm would then also be applicable to the DeterministicReplication problem. In the next section, we present such an algorithm.
CHAPTER 3. CENTRALIZED ALGORITHM

3.3 Approximation Algorithm

In this section, we present an approximation algorithm for the MultiPartialSetCover problem, which is also applicable to the DeterministicReplication problem.

We do not devise directly an approximation algorithm for the MultiPartialSetCover problem but for a generalization of the MultiPartialSetCover problem. In this generalized problem, coined WeightedMultiPartialSetCover, each set has a cost and the goal is to find a $\varphi$-partial cover of $U$ with minimum cost. This generalization appears natural since the PartialSetCover problem (see Section 3.1) is usually described as a weighted version in the literature (e.g. [Sla97, GKS04]). Although we only use uniform costs to solve our problem, a weighted problem version (together with an approximation algorithm and an approximation guarantee) might be useful to consider further constraints such as storage or load capacity constraints of peers or other optimization criteria such as network traffic. Moreover, the WeightedMultiPartialSetCover problem might have other applications beyond the scope of P2P networks.

By extending Slavík’s definition of the partial set cover problem with set costs [Sla97], we obtain the following definition of the WeightedMultiPartialSetCover problem.

Let $U = \{U_1, U_2, \ldots, U_{n'}\}$ be a family of finite and pairwise disjoint ground sets. Let $S = \{S_1, S_2, \ldots, S_n\}$ be a family of subsets of $\bigcup_{t=1}^{n'} U_t$, such that $S$ is a cover of $\bigcup U_t$ (that is $\bigcup S_j = \bigcup U_t$). Let $c = \{c_1, c_2, \ldots, c_n\}$ be positive real costs of each set in the cover. The cover $S$ is identified with the set of indices $J = \{1, 2, \ldots, n\}$ and any subset $S^* \subseteq S$ is identified with the corresponding set $J^* \subseteq J$. Let $m_t = |U_t|$ for all $t = 1, \ldots, n'$ and define $m = n_1 + \cdots + m_{n'}$. Let $\varphi = \{p_1, p_2, \ldots, p_{n'}\}$ with $0 \leq p_t \leq 1$ for all $t = 1, \ldots, n'$. We say that $J^*$ or alternatively $S^*$ is a $\varphi$-partial cover of $U$ if

$$|U_t \cap \bigcup_{j \in J^*} S_j| = |U_t \cap \bigcup_{j \in S} S_j| \geq p_t m_t$$

for all $t = 1, \ldots, n'$.

Let $c(J^*) = c(S^*) = \sum_{j \in J^*} c_j$ be the cost of the $\varphi$-partial cover $J^*$ ($S^*$) and define $c_{min}$ as the minimum possible cost of any $\varphi$-partial cover of $U$.

**Partial set cover problem with multiple ground sets and set costs (WeightedMultiPartialSetCover):** Given a family of finite and pairwise disjoint ground sets $U = \{U_1, U_2, \ldots, U_{n'}\}$, a finite cover $S$ of $\bigcup U_t$, positive costs $c$, $0 \leq p_t \leq 1$ for all $t = 1, \ldots, n'$, the goal is to find a $\varphi$-partial cover $J^*$ of $U$ with minimum cost.

In our P2P setting (DeterministicReplication problem), we have $n' = n$ and $c_1 = \cdots = c_n = 1$. 
Algorithm 1 Centralized greedy algorithm \((U, S, c, \phi)\)

1: \(J^* \leftarrow \emptyset\)
2: \(S_j^{(0)} \leftarrow S_j\) for each \(j = 1, \ldots, n\)
3: while true do
4: \(r_t \leftarrow \max(0, \ceil{p_t m_t} - |U_t \cap \bigcup_{j \in J^*} S_j^{(0)}|)\) for each \(t = 1, \ldots, n'\)
5: if \(r_t = 0\) for all \(t = 1, \ldots, n'\) then
6: output \(J^*\)
7: STOP
8: end if
9: Let \(u_j = \sum_{t=1}^{n'} \min(r_t, |U_t \cap S_j|)\). Find \(i \in J \setminus J^*\) that maximizes the quotient \(u_j/c_j\) for \(j \in J \setminus J^*\). In case of a tie, take the smallest such \(i\).
10: \(J^* \leftarrow J^* \cup \{i\}\)
11: \(S_j \leftarrow S_j \setminus S_i\) for each \(j \in J\)
12: end while

For the WeightedMultiPartialSetCover problem, we propose a greedy approximation algorithm (see Algorithm 1). This algorithm extends Slavík’s approximation algorithm [Sla97] proposed for the partial set cover problem with set costs. Note that Slavík’s algorithm is contained as a special case in our algorithm for \(n' = 1\).

At each step, the algorithm computes for each set \(S_j\) the number of relevant elements \(u_j\) (line 9 of the algorithm), which is the number of uncovered elements that contribute towards achieving a \(\phi\)-partial cover. To compute this number, the algorithm first computes \(r_t\) for each \(t = 1, \ldots, n'\) (line 4), which is the number of elements in each \(U_t\) that yet need to be covered to in order obtain a \(\phi\)-partial cover. The algorithm then adds the set with the highest number of relevant elements per cost unit \((u_j/c_j)\) to the current solution. Finally, the algorithm stops as soon as a \(\phi\)-partial cover is found. The algorithm is guaranteed to terminate because at each step it adds one set to the current solution and the cover \(S\) is a \(\phi\)-partial cover of \(\bigcup U_t\).

For each set \(S_j\), we define the number of relevant elements per cost unit as the **worthiness** of \(S_j\) to be included in the greedy cover. The worthiness of a set can also be regarded as an estimation of the probability that this set is included in a minimum partial cover.

In the P2P context, sets correspond to peers and the \(\phi\)-partial cover produced by the centralized algorithm corresponds to a replica allocation satisfying the availability constraints. Ties are then broken in line 9 of the algorithm by choosing the peer with the smallest id. A peer has a worthiness to be included in the replica allocation computed by the centralized algorithm. More precisely, the worthiness of a peer \(v_i\) is the number of unsuccessful query executions that sample \(v_i\) and
CHAPTER 3. CENTRALIZED ALGORITHM

contribute towards achieving the availability constraints. Hence, the worthiness of a peer is determined by the replica assignments previously chosen by the centralized algorithm. In Chapter 4, we use this property to turn the centralized algorithm into a distributed one.

Figure 3.1 shows three iterations of the centralized algorithm in a P2P scenario. In this example, four peers (shown by colored rings around black circles) issue queries for the same data item in a network. The corresponding query executions are shown as arrows. The number in each box denotes the $r_t$ value of a peer, that is, the number of query executions initiated by this peer that yet need to be satisfied in order to satisfy its availability constraints. Moreover, black numbers denote the worthiness of peers. Peers without such a number have a worthiness of zero. Figure 3.1a shows the first iteration of the algorithm. For instance, the peer in the center with a worthiness of three is sampled by two query executions with red square dotted lines and two query executions with green solid lines. Both of the query executions with solid lines contribute to the worthiness of this peer. In contrast, only one of the two query executions with square dotted lines contribute to its worthiness since only one of these two query executions needs to be successful to satisfy the availability constraint of their initiator. The centralized algorithm chooses the peer with the highest worthiness (4) and assigns a replica to this peer. Figure 3.1b shows the second iteration of the centralized algorithm with updated $r_t$ and worthiness values. The previous replica assignment is shown with a white fill. Moreover, the figure does not show query executions that are now successful or not relevant any more because the availability constraint of their query initiator is satisfied. Again, the distributed algorithm assigns a replica to the peer with highest worthiness (3). Figure 3.1c shows the third and last iteration of the centralized algorithm where all $r_t$ values are zero and hence all availability constraints are satisfied. The two resulting replica assignments are the output allocation of the centralized algorithm.

Finally, we consider the runtime complexity of the centralized algorithm. Since the algorithm adds at most $n$ sets to the output $\varphi$-partial cover and in each iteration at least one more relevant element is covered, we have at most $\min(n, \sum_{t=1}^{n'} [pm_t])$ loop iterations. In the loop body, each of the operations (i) computing $r_t$ for each ground set (line 4 of the algorithm), (ii) computing $u_j$ for each set $S_j$ (line 9), and (iii) removing the elements of the selected set $S_i$ from all sets $S_j$ (line 11) can easily be implemented to run in time $O(nm)$. This can be achieved using a (0,1)-matrix representing the elements of the initial sets $S_j^{(0)}$ and a (0,1)-matrix representing the uncovered elements of the sets $S_j$. While the former matrix is needed to compute $r_t$ for each ground set $U_t$, the latter matrix is required to compute $u_j$ for each set $S_j$ and is modified during each iteration. Note that both matrices coincide at the beginning of the algorithm. Table 3.1
Figure 3.1: Centralized algorithm in a P2P setting
Table 3.1: Example of a matrix representing the sets $S_j$

<table>
<thead>
<tr>
<th>Elements</th>
<th>$U_1$</th>
<th>$U_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>1 0 1</td>
<td>0 0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>1 1 0</td>
<td>0 1</td>
</tr>
<tr>
<td>$S_3$</td>
<td>0 0 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>

shows an example of such a matrix (at the beginning of the algorithm) for a family of ground sets $U = \{U_1, U_2\}$ with $U_1 = \{1, 2, 3\}$ and $U_2 = \{4, 5\}$, and a family of subsets $S = \{S_1, S_2, S_3\}$ with $S_1 = \{1, 3\}$, $S_2 = \{1, 2, 5\}$, and $S_3 = \{3, 4, 5\}$. Hence, there is an implementation of the algorithm that runs in time

$$O(\min(n, \sum_{t=1}^{n'} \lceil p_t m_t \rceil) \cdot nm).$$

Note that there might be more efficient implementations of the centralized algorithm with lower runtime complexity.

### 3.4 Approximation Guarantee

In this section, we prove an approximation guarantee for the centralized approximation algorithm presented in the previous section. More precisely, we prove an approximation ratio, which is also called performance ratio. For minimization problems, an approximation ratio is an upper bound on the ratio between the cost of the approximation solution and the cost of the optimum solution (see Cormen et al. [CLRS01], for instance). In our P2P setting, a solution is a replica allocation $A_j$ and the cost of a solution is the number of replicas $|A_j|$. The following proof is based on the proof given by Slavík [Sla97]. For the partial set cover problem with set costs, Slavík [Sla97] proposes a greedy approximation algorithm (see also the previous section) and proves an approximation ratio. The proof given below follows the line of arguments in Slavík’s proof and generalizes arguments and properties of Slavík’s algorithm to our generalized algorithm. We finally obtain the same approximation ratio for our approximation algorithm as Slavík proved for his algorithm.

For proving an approximation ratio, we further constrain the WeightedMultiPartialSetCover problem by assuming that there is at least one ground set $U_t \in U$ that is non-empty and for which $p_t > 0$. This is to ensure that for each instance of the WeightedMultiPartialSetCover problem, we have $c_{\min} > 0$ and hence a well defined approximation ratio. Denote by $c_{\text{greedy}}$ the cost of the $\varphi$-partial cover.
produced by the centralized greedy algorithm. Note that in case no elements need to be covered, we have \( c_{\text{min}} = c_{\text{greedy}} = 0 \).

Let \( \{O_1, O_2, \ldots, O_\delta\} \) be a \( \varphi \)-partial cover of \( U \) \( (O_s \in S \text{ for all } s = 1, \ldots, \delta) \) with minimum cost \( c_{\text{min}} \). Denote the costs of this minimum cover by \( \alpha_1, \alpha_2, \ldots, \alpha_\delta \). Hence, \( c_{\text{min}} = \alpha_1 + \alpha_2 + \cdots + \alpha_\delta \). Without loss of generality, we can assume that the sets \( O_1, \ldots, O_\delta \) are pairwise disjoint and that

\[
|U_t \cap \bigcup_{s=1}^\delta O_s| = \lceil pt m_t \rceil \text{ for all } t = 1, \ldots, n'.
\]  

(3.1)

This can be achieved by deleting elements from the sets \( O_1, \ldots, O_\delta \) as follows. For each set \( O_s \) that is chosen by the greedy algorithm, all elements are deleted from \( O_s \) that were not relevant (i.e. that were already covered or did not contribute towards achieving a \( \varphi \)-partial cover) in the iteration of the greedy algorithm when \( O_s \) was chosen. For each set \( O_s \) that is not chosen by the greedy algorithm, elements are arbitrarily deleted to satisfy the constraints above. In this way, each possibly modified set \( O_s \) that was previously chosen by the greedy algorithm has the same number of relevant elements as before at each iteration of the greedy algorithm. Moreover, for each set \( O_s \) not chosen by the greedy algorithm, the number of relevant elements may only be decreased. Hence, the greedy algorithm produces the same output as before. The sets \( O_1, \ldots, O_\delta \) remain a \( \varphi \)-partial cover; hence, the value of the optimal solution does not increase. Then

\[
\sum_{s=1}^\delta |O_s| = \sum_{t=1}^{n'} \lceil pt m_t \rceil. \tag{3.2}
\]

Let \( d \) be the number of sets in the \( \varphi \)-partial cover produced by the greedy algorithm. We order the sets in the cover \( S \) such that the greedy algorithm chooses index \( i \) in its \( i \)-th iteration, that is, \( c_{\text{greedy}} = c_1 + c_2 + \cdots + c_d \).

Let \( r_i^{(i)} \) be the number of elements in \( U_t \) remaining to be covered after the \( i \)-th iteration of the greedy algorithm and let \( r^{(i)} = \sum_{i=1}^{n'} r_i^{(i)} \), i.e. \( r^{(i)} \) is the total number of elements remaining to be covered after the \( i \)-th iteration. Let \( O_s^{(i)} \) and \( S_j^{(i)} \) denote the sets \( O_s \) and \( S_j \) after the \( i \)-th iteration. Define \( u_j^{(i)} = \sum_{i=1}^{n'} \min(r_i^{(i)}, |U_t \cap S_j^{(i)}|) \) and \( a_s^{(i)} = \sum_{i=1}^{n'} \min(r_i^{(i)}, |U_t \cap O_s^{(i)}|) \). Hence, \( u_j^{(i)} (a_s^{(i)}) \) is the number of relevant elements of the set \( S_j \) (\( O_s \)) after the \( i \)-th iteration of the greedy algorithm.

In the following, we consider all fractions

\[
\frac{k_s}{\alpha_s}, \quad s = 1, \ldots, \delta, \quad k_s = 1, \ldots, |O_s|.
\]
By equation (3.2), we have \( \sum_{s=1}^{\delta} |O_s| = \sum_{t=1}^{n'} [p_t m_t] = r^{(0)} \)-many fractions. If we rearrange these fractions into a non-decreasing sequence \( e_1 \leq e_2 \leq \cdots \leq e_r^{(0)} \) then the following inequality holds.

**Lemma 1.** For each \( i = 0, \ldots, d - 1 \), we have

\[
\frac{u^{(i)}_{i+1}}{c_{i+1}} \geq e_{r^{(i)}}.
\]

**Proof.** Without loss of generality, we assume that in line 11 of Algorithm 1 not all elements of \( S_i \) are removed from each set \( S_j \) but only a subset \( T \subseteq S_i \) of exactly \( u_i \) elements that were previously uncovered and contribute towards achieving a \( \varphi \)-partial cover. Removing such a subset \( T \) from each set \( S_j \) only prevents from deleting more elements than necessary and hence the modified algorithm produces the same result as before.

After the \( i \)-th iteration of the greedy algorithm, \( i = 0, \ldots, d - 1 \), there are exactly \( r_t^{(i)} \) elements to be covered in each \( U_t \in U \). With the previous assumption, for all \( t = 1, \ldots, n' \) there are exactly \( [p_t m_t] - r_t^{(i)} \) elements deleted in \( U_t \) in previous steps and hence there are at most \( [p_t m_t] - r_t^{(i)} \) elements deleted in \( U_t \cap \bigcup O_s \subseteq U_t \) in previous steps. Since \( |U_t \cap \bigcup O_s| = [p_t m_t] \) for all \( t = 1, \ldots, n' \) by equation (3.1), there are at least \( r_t^{(i)} \) elements in \( U_t \cap \bigcup O_s \) not deleted in previous steps for all \( t = 1, \ldots, n' \). Hence, \( |U_t \cap \bigcup O_s^{(i)}| \geq r_t^{(i)} \) for all \( t = 1, \ldots, n' \).

Since we assume that all \( O_s \) are pairwise disjoint, all \( O_s^{(i)} \) are also disjoint and hence

\[
\sum_{s=1}^{\delta} |U_t \cap O_s^{(i)}| = |U_t \cap \bigcup_{s=1}^{\delta} O_s^{(i)}| \geq r_t^{(i)} \text{ for all } t = 1, \ldots, n'. \tag{3.3}
\]

Moreover, we have

\[
\sum_{s=1}^{\delta} min(r_t^{(i)}, |U_t \cap O_s^{(i)}|) \geq r_t^{(i)} \text{ for all } t = 1, \ldots, n'
\]

because if \( |U_t \cap O_s^{(i)}| < r_t^{(i)} \) for all \( s = 1, \ldots, \delta \) then the above inequality holds by equation (3.3). Otherwise, the inequality holds trivially. Therefore,

\[
\sum_{s=1}^{\delta} a_s^{(i)} = \sum_{s=1}^{\delta} \sum_{t=1}^{n'} \min(r_t^{(i)}, |U_t \cap O_s^{(i)}|)
\]

\[
= \sum_{t=1}^{n'} \sum_{s=1}^{\delta} \min(r_t^{(i)}, |U_t \cap O_s^{(i)}|)
\]

\[
\geq \sum_{t=1}^{n'} r_t^{(i)}
\]
and since \( r^{(i)} = \sum_{t=1}^{n'} r_t^{(i)} \), we have
\[
\sum_{s=1}^{\delta} d_s^{(i)} \geq r^{(i)}.
\] (3.4)

In every iteration \((i + 1)\) of the greedy algorithm, a subscript \( j \in J \setminus J^* \) is chosen for which \( u_j^{(i)}/c_j \) is maximal. Since \( u_j^{(i)} = 0 \) for each \( j \in J^* \), we have
\[
\frac{u_{i+1}^{(i)}}{c_{i+1}} \geq \frac{u_j^{(i)}}{c_j},
\]
for all \( j \in J \). Hence, in particular,
\[
\frac{u_{i+1}^{(i)}}{c_{i+1}} \geq \frac{d_s^{(i)}}{\alpha_s},
\]
for all \( s = 1, \ldots, \delta \). Therefore,
\[
\frac{u_{i+1}^{(i)}}{c_{i+1}} \geq \frac{k_s}{\alpha_s},
\]
for all \( s = 1, \ldots, \delta \) and all \( k_s = 1, \ldots, d_s^{(i)} \), which are \( \sum_{s=1}^{\delta} d_s^{(i)} \)-many fractions \( k_s/\alpha_s \). Hence,
\[
\frac{u_{i+1}^{(i)}}{c_{i+1}} \geq e_j
\]
for at least \( \sum_{s=1}^{\delta} d_s^{(i)} \)-many indices \( j \) and by equation (3.4) also for at least \( r^{(i)} \) indices \( j \). Let \( h \) be the largest such index. Since there are at least \( r^{(i)} \) such indices \( j \), we have \( h \geq r^{(i)} \). Moreover, since \( e_1 \leq \cdots \leq e_{r^{(i)}} \), we have
\[
\frac{u_{i+1}^{(i)}}{c_{i+1}} \geq e_h \geq \cdots \geq e_{r^{(i)}} \geq \cdots \geq e_1.
\]
Hence,
\[
\frac{u_{i+1}^{(i)}}{c_{i+1}} \geq e_{r^{(i)}} \text{ for any } i = 0, \ldots, d - 1.
\]

From Lemma 1, we easily obtain:

**Lemma 2.** We have
\[
c_1 + \cdots + c_d \leq \alpha_1 H(|O_1|) + \cdots + \alpha_\delta H(|O_\delta|),
\]
where \( H(\mu) = 1 + \cdots + 1/\mu \) is the \( \mu \)th harmonic number for all \( \mu \in \mathbb{N} \).
Proof. By Lemma 1, we have
\[ \frac{u_r^{(i)}}{c_{i+1}} \geq e_{r(i)} \geq e_{r(i)-1} \geq \cdots \geq e_{r(i+1)+1}. \]
Since \( r^{(i)} - r^{(i+1)} = u_r^{(i)} \) and \( 1/e_{r(i)} \leq 1/e_{r(i)-1} \leq \cdots \leq 1/e_{r(i+1)+1} \), we have
\[ c_{i+1} \leq \frac{u_r^{(i)}}{e_r^{(i)}} \cdot \frac{1}{e_r^{(i)}} + \frac{1}{e_r^{(i)-1}} + \cdots + \frac{1}{e_r^{(i+1)+1}}. \]
By adding the above inequalities for \( i = 0, \ldots, d-1 \), we obtain
\[ c_1 + \cdots + c_d \leq \frac{1}{e_r^{(0)}} + \cdots + \frac{1}{e_1} = \alpha_1 H(|O_1|) + \cdots + \alpha_\delta H(|O_\delta|). \]
Note that \( r^{(d)} = 0 \) and hence \( e_{r(\delta)+1} = e_1 \) since the greedy output consists of \( d \) sets and thus there are no more elements to be covered after \( d \) iterations. \( \square \)

We now easily obtain the following two theorems.

**Theorem 1.** Let \( \theta \in \mathbb{N} \) such that \( |S_j| \leq \theta \) for all \( j = 1, \ldots, n \). The centralized greedy algorithm achieves an approximation ratio of \( H(\theta) \) for the WeightedMultiPartialSetCover problem, where \( H(\mu) = 1 + \cdots + 1/\mu \) is the \( \mu \)th harmonic number.

**Proof.** By Lemma 2 and the fact that \( H(|O_s|) \leq H(\theta) \) for all \( s = 1, \ldots, \delta \), we have
\[ c_{\text{greedy}} = c_1 + \cdots + c_d \leq \alpha_1 H(|O_1|) + \cdots + \alpha_\delta H(|O_\delta|) \leq H(\theta) \cdot (\alpha_1 + \cdots + \alpha_\delta) = H(\theta) \cdot c_{\text{min}}. \]
Hence, \( c_{\text{greedy}}/c_{\text{min}} \leq H(\theta) \). \( \square \)

By Equation (3.2), we have \( |O_s| \leq \sum_{s=1}^\delta |O_s| = \sum_{t=1}^{n'} [p_t m_t] \) for all \( s = 1, \ldots, \delta \). Together with Lemma 2, this proves that \( c_{\text{greedy}}/c_{\text{min}} \leq H(\sum_{t=1}^{n'} [p_t m_t]) \). Hence, we obtain the following theorem.

**Theorem 2.** The centralized greedy algorithm achieves an approximation ratio of \( H(\sum_{t=1}^{n'}[p_t m_t]) \) for the WeightedMultiPartialSetCover problem.
In the P2P context, $\theta$ is an upper bound for the number of query executions that sample a peer, whereas $\sum_{t=1}^{n'} [p_t m_t]$ is the total number of query executions that need to be successful in order to satisfy the availability constraints. Note that Theorem 2 is stronger than Theorem 1 in case $\sum_{t=1}^{n'} [p_t m_t] < \theta$. Moreover, the approximation is logarithmically bounded since $H(\mu) \leq \ln(\mu) + 1$ for all $\mu \in \mathbb{N}$.

Finally, we show that the approximation bounds of Theorem 1 and Theorem 2 are tight. To this end, we extend an example given by Chvátal [Chv79] and modified by Slavík [Sla97].

**Example 1.** Given $U = \{U_1\}$ with $U_1 = \{1, \ldots, m\}$ and $0 < p_1 \leq 1$ we construct a cover of $\bigcup_{i=1}^{n'} U_i = U_1$ as follows. Let $u = \sum_{t=1}^{n'} [p_t m_t] = \lfloor p_1 m_1 \rfloor$. Define $S_i = \{i\}$ for $i = 1, \ldots, m$ and set $S_{m+1} = \{1, \ldots, u\}$. Define $c_i = u/(u - i + 1)$ for $i = 1, \ldots, u$, and $c_i = u$ for $i = u + 1, \ldots, m + 1$. Then $J^* = \{1, \ldots, u\}, c_{\text{greedy}} = \frac{u}{u} + \cdots + \frac{u}{1} = uH(u)$, and $c_{\text{min}} = c_{m+1} = u$. Thus, $c_{\text{greedy}}/c_{\text{min}} = H(\sum_{t=1}^{n'} [p_t m_t]) = H(\theta)$.

It would be interesting to investigate whether the approximation bounds are also tight for the centralized algorithm when solving the DeterministicReplication problem. The previous example cannot straightforwardly be used to prove this since the example uses heterogeneous set costs, whereas we use uniform set costs for solving the DeterministicReplication problem.

### 3.5 Issues in P2P Networks

In the previous section, we proved an approximation guarantee for the centralized algorithm. More precisely, we proved that the approximation is bounded logarithmically on the maximum number of query executions in $Q$ that sample a peer and bounded logarithmically on the number of query executions in $Q$ that need to be successful in order to satisfy the availability constraints. Both the maximum number of query executions that sample a peer and the number of query executions that need to be successful are limited by the total number of query executions $|Q|$. Hence, a way to achieve a desired approximation guarantee is to limit the size of $Q$.

With a limited number of query executions, it may only be possible to estimate the parameters of the ProbabilisticReplication problem. Therefore, we have a trade-off between the quality of parameter approximation and the ratio of the approximation bound. However, increasing the size of $Q$ does not necessarily decrease the “average” approximation quality of the centralized algorithm in P2P settings. In fact, our experiments (see next section) show that the approximation quality does not degrade for an increasing number of queries.
Another issue is that our centralized approximation algorithm cannot be used directly in P2P networks because the algorithm requires global knowledge about query executions in the network. Moreover, it is desirable to have a distributed algorithm running on each peer to improve performance and robustness. Since a replica allocation can be computed for each data item independently, this could be exploited by running the algorithm for each data item in parallel on different peers. However, this would require again that individual peers have complete knowledge of the network and it would require assigning responsibilities for computing replica allocations to individual peers. Hence, it is desirable for each data item to compute a replica allocation in a distributed way with only local knowledge. In Chapter 4, we present a distributed algorithm that satisfies these requirements.

3.6 Experiments on the Approximation Quality

In this section, we present an experimental evaluation of the approximation algorithm proposed in Section 3.3. The goal of this evaluation is to investigate the approximation quality of the algorithm in different P2P settings. To this end, we performed several simulations with random graphs. In Section 3.6.1, the simulation setup and the implementation of the simulation is described. In Section 3.6.2, the simulation results are presented and discussed.

3.6.1 Simulation Setup and Implementation

In order to generate network topologies for our simulation, we use the Erdős-Rényi model [ER60] to construct random graphs. In this model, each of the possible \( \binom{n}{2} \) edges in a graph with \( n \) vertices occurs independently with probability \( p \). Moreover, since the DeterministicReplication problem can be solved for each data item independently, we only consider a single data item \( j \) in this evaluation.

For the simulation of search in the network, we use a modified version of the stochastic process used for the ProbabilisticReplication problem (see Section 2.1). In our modified process, at each step exactly one query is issued by a peer that is chosen uniformly at random from all online peers. This modification makes it straightforward to issue a fixed number of queries during a simulation run. As in the original process, at each step each peer \( v_i \) is online independently with probability \( o_i \) and while a query message is forwarded, the online/offline states of peers do not change.

The query rate of a peer \( v_i \) is defined as the probability that \( v_i \) issues a query in a step, given that \( v_i \) is online (see Section 2.1). Hence, the query rate of \( v_i \) is the
Table 3.2: Default parameter values for evaluating the approximation quality

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of peers</td>
<td>20</td>
</tr>
<tr>
<td>Edge probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Number of queries issued in the network</td>
<td>500</td>
</tr>
<tr>
<td>Random walk length</td>
<td>4</td>
</tr>
<tr>
<td>Availability constraint of each peer</td>
<td>0.8</td>
</tr>
<tr>
<td>Online probability of each peer</td>
<td>0.8</td>
</tr>
</tbody>
</table>

quotient between the probability that \(v_i\) issues a query in a step, which implies that \(v_i\) is online, and the probability that \(v_i\) is online in a step. (The conditional probability of event \(A\) given event \(B\) is defined by \(\Pr[A|B] = \Pr[A \cap B] / \Pr[B]\) if \(\Pr[B] > 0\); see Wasserman [Was04], for instance). Since we use the same (positive) online probability \(o_i\) for each peer \(v_i\) in our experiments, we also have the same probability for each peer to issue a query in a step of the modified stochastic process. Thus, we have uniform query rates in our experiments.

After issuing a certain number of queries in the network, the optimum and the approximation solution are computed to evaluate the quality of the approximation solution. In order to determine which parameters affect the approximation quality, we vary each of the following parameters separately:

- Edge probability (probability for each possible edge to occur in the random graph)
- Number of queries issued in the network
- Random walk length
- Availability constraint of each peer
- Online probability of each peer

Parameters not varied during a simulation run are assigned a default value (see Table 3.2). Moreover, we perform 100 simulation runs for each parameter configuration. At each run, a new random graph and a new set of queries is generated.

In our experiments, we have the same availability constraint and the same online probability for each peer. Clearly, it would also be interesting to perform simulations with heterogeneous query rates, online probabilities, and availability constraints.

The optimum solution is computed using a backtracking algorithm (see Brassard and Bratley [BB96], for instance) with exponential runtime on the number of
peers. This algorithm explores the search space by incrementally adding peers to a candidate replica allocation and removing peers from this allocation when tracking back. Testing whether a candidate allocation satisfies the availability constraints is performed on the fly whenever a peer is added to the candidate allocation. The algorithm only adds another peer to a candidate allocation if the resulting allocation would contain fewer replicas than the best solution found so far. This implies that further peers are only added to candidate allocations that do not satisfy the availability constraints yet. Moreover, a new peer is only added to a candidate allocation if creating a replica at this peer satisfies at least one query execution that contributes towards achieving the availability constraints. With these optimizations, we were able to compute the optimum solution for up to 20 peers within acceptable time.

3.6.2 Results and Discussion

Figures 3.2-3.7 show how the different parameters affect the optimum and the approximation solution in terms of the required number of replicas. Over all sets of 100 runs, the sample variance (see Wasserman [Was04], for instance) of the optimum solution is at most 4.65 and the sample variance of the approximation solution is at most 5.07.

Our main observation is that the number of replicas required by the approximation solution is always very close to the number of replicas required by the optimum solution, regardless of the parameter being varied. In the aggregated values (average over 100 runs), the ratio between approximation and optimum solution is at most 1.141 and the absolute difference between both solutions is at most 1.550.

Figure 3.2 shows the number of required replicas for the optimum and the approximation solution as the edge probability increases. If the edge probability is zero then there are no edges at all in the graph and hence query messages cannot be forwarded to any neighbors. Then a replica needs to be assigned to each peer that issues at least one query. The number of replicas decreases with an increasing edge probability because a random walk traverses more peers then and replicas satisfy a larger number of queries. However, this effect gradually slows down and finally stops with an edge probability of 0.35 since the number of peers that are sampled by a query execution does then not significantly increase any more. Interestingly, starting with an edge probability of 0.35, the number of replicas increases slightly with an increasing edge probability. This might be due to the fact that with a lower edge probability there is more heterogeneity in the degrees of peers (number of neighbors) and random walks of query executions are
more likely to go through some few peers with high degree. Such peers are then good replica candidates because they are sampled by many query executions.

Figures 3.3 and 3.4 show the results for an increasing number of queries issued in the network. Figure 3.3 shows the results for issuing up to 100 queries. If no queries are issued at all in the network, the availability constraints of all peers are trivially satisfied and hence no replicas need to be created. When the number of issued queries increases, the number of peers that issue at least one query increases, requiring more replicas to be created. This effect lasts for up to 70 queries, when it is quite likely that each peer issues at least one query. Figure 3.3 shows the number of required replicas for up to 8000 queries. When the number of issued queries increases, the expected maximum number of query executions that sample a peer increases and hence the approximation bound (i.e. the approximation ratio) also increases. However, we can observe that the actual approximation quality does not degrade.
CHAPTER 3. CENTRALIZED ALGORITHM

The results for an increasing random walk length are shown in Figure 3.5. For a walk length of zero, query messages are not forwarded and hence query executions sample only the query initiator. A replica is then required for every peer that issues at least one query, which is similar to an edge probability of zero in Figure 3.2. When the walk length is increased, a query execution samples more peers and hence fewer replicas are required. However, this effect slows down with an increasing walk length since there are only a limited number of peers in the network and it becomes more likely that query executions sample the same peers multiple times. Interestingly, in this small network with 20 peers, the number of required replicas for a walk length of one is already close to the number of required replicas for larger walk lengths. Note that it is possible to have a walk length greater than the number of peers in the network. In this case, the query message circulates in the network.

Figure 3.4: Approximation results for up to 8000 queries

Figure 3.5: Approximation results for different walk lengths
Figure 3.6 shows the number of replicas for the optimum and the approximation solution as the availability constraint for each peer increases. If all availability constraints are zero, the availability constraints are trivially satisfied. For slightly larger availability constraints, at least one query needs to be successful for each peer that issues at least one query. This explains the sudden increase of required replicas. When the availability constraints further increase, the number of replicas that are required to satisfy the availability constraints naturally also increases.

Figure 3.7 shows the results as the online probability of each peer increases. If the online probability is close to zero then only few peers (if any) are online at a time and it is unlikely for these peers to be connected. Then, a query execution cannot sample other peers apart from the query initiator and hence a replica is needed for each peer that issues at least one query. This is similar to an edge probability of zero and to a walk length of zero. The higher the online probability of each
peer, the more peers are sampled on average by a query execution. Replicas can then satisfy more queries, which reduces the number of required replicas.
Chapter 4

Distributed Algorithm

In this chapter, we turn the centralized algorithm for the deterministic replication problem (DeterministicReplication) into a distributed one by exploiting local properties of the centralized algorithm. We prove that the distributed algorithm is equivalent to the centralized algorithm, meaning that for any given set of query executions, the distributed algorithm produces the same replica allocation for each data item as the centralized algorithm. Moreover, we experimentally evaluate the efficiency of the distributed algorithm in terms of the time needed to construct replica allocations.

4.1 Input of the Distributed Algorithm

In the previous chapter, we devised a centralized algorithm that takes a set of query executions as input. An open problem is how to obtain such a set of query executions. One approach is to estimate the parameters of the stochastic process described in Section 2.1 and to compute for each item \( j \) and for each possible walk \( W \) of length \( l \) the probability \( r_{W,j} \) that a query issued by the stochastic process for item \( j \) goes through \( W \) during its execution. These \( r_{W,j} \) probabilities can then be transformed into a set of query executions \( Q \) that sufficiently approximates these probabilities. However, it is not straightforward to compute these \( r_{W,j} \) probabilities in a distributed way.

We therefore pursue another approach to obtain a set of query executions \( Q \), which is straightforward to implement in a distributed way. For a certain time period, each peer keeps track of the query executions it initiates and the query
executions by which it is sampled. In this way, for each peer $v_i$ and each item $j$, we obtain a set $Q_{ij}$ of query executions initiated by peer $v_i$ for item $j$ and a set $R_{ij}$ of query executions for item $j$ that sample peer $v_i$. For $Q_j$, the set of all query executions for item $j$, we have $Q_j = \bigcup_{i=1}^{n} Q_{ij} = \bigcup_{i=1}^{n} R_{ij}$ and for the set of all query executions $Q$, we have $Q = \bigcup_{j=1}^{k} Q_j$. This approach makes an estimation of the stochastic parameters unnecessary. In fact, the parameters of the stochastic process are implicitly estimated in this approach.

If the network is in a steady state, the set of obtained query executions increases over time, which continuously improves the parameter approximations of the underlying stochastic processes. Note that this approach for obtaining a set of query executions $Q$ is independent of the underlying stochastic model and relies only on stable query patterns and stable online/offline behavior.

For the DeterministicReplication problem (see Section 2.2), we assumed that no data items are stored at the peers in $V$. The above-presented approach for obtaining a set of query executions $Q$ can also be pursued in a network where replicas have already been placed. It is then necessary that forwarding a query message does not stop when the random walk of the query execution traverses a peer that stores the requested item. This can also be regarded as emulating an empty network. In the following, we therefore assume that random walks have a fixed length. Clearly, having walks of fixed length incurs some overhead and implies that a query is sent through the network even if the requested data item is locally available.

### 4.2 Local Properties of the Centralized Algorithm

In this section, we describe local properties of the centralized algorithm that enable us to turn the centralized algorithm into a distributed one.

In the following, we use the notion of $l$- and $2l$-neighborhoods, where $l$ denotes the (fixed) length of random walks. The $l$-neighborhood of a peer $v_i$ is the set of peers that are at most $l$ hops away from $v_i$. These are also the peers that can be sampled by query executions initiated by peer $v_i$. Moreover, the $2l$-neighborhood of a peer $v_i$ is the set of peers that are at most $2l$-hops away from $v_i$. Figure 4.1 shows the $l$ and the $2l$-neighborhood of a peer in an example network for $l = 2$.

The distributed algorithm is devised based on the following local properties of the centralized greedy algorithm and the DeterministicReplication problem.

- The availability constraints of a peer $v_i$ can only be satisfied by replicas placed on peers in the $l$-neighborhood of $v_i$ since only these peers can be sampled by queries executions initiated by $v_i$. Hence, each peer can deter-
mine with local knowledge whether its availability constraints are satisfied or not.

- Recall that in a P2P setting, the worthiness of a peer \( v_i \) for an item \( j \) is the number of query executions for item \( j \) that sample this peer and contribute in satisfying the availability constraints of the peers that initiated these queries executions (see Section 3.3). Since query executions initiated by peers in the \( l \)-neighborhood of \( v_i \) can only reach peers in the \( 2l \)-neighborhood of \( v_i \), the worthiness of a peer \( v_i \) for an item \( j \) is determined by the replica assignments of item \( j \) to peers in the \( 2l \)-neighborhood of \( v_i \). Hence, each peer can compute its own worthiness for each data item with local knowledge about its \( 2l \)-neighborhood. Note that we do not further investigate methods for peers to efficiently compute their own worthiness because we are primarily interested in the principles of a distributed algorithm.

- The centralized algorithm chooses at each step the peer with highest worthiness. Moreover, only replica assignments to peers in the \( 2l \)-neighborhood of a peer \( v_i \) can affect the worthiness of \( v_i \) for any data item (see above). These properties of the centralized algorithm imply the following. If a peer has the highest worthiness in its \( 2l \)-neighborhood for an item \( j \) at any step...
of the centralized algorithm then the centralized algorithm cannot assign a replica of $j$ to any peer in the $2l$-neighborhood of $v_i$, unless the algorithm first assigns a replica of $j$ to $v_i$. (Later, we also consider the case that multiple peers may have the same highest worthiness at a time.) Hence, if a peer $v_i$ has the highest worthiness of all the peers in its $2l$-neighborhood at any step of the centralized algorithm then from all the peers in the $2l$-neighborhood of $v_i$, the centralized algorithm must first assign a replica to $v_i$. This property is formally stated and proven as Lemma 3 in Section 4.4.

### 4.3 Algorithm

In this section, we turn the centralized algorithm into a distributed one by exploiting the local properties of the centralized algorithm described in the previous section. Moreover, we exploit the property of the centralized algorithm that the worthiness of a peer can be computed for any given data item $j$ and any given replica allocation $A_j \subseteq V$, regardless of whether this allocation occurs in an iteration of the centralized algorithm or not.

Let $n_l : V \rightarrow \mathcal{P}(V)$, such that $n_l(v_i)$ evaluates to the $l$-neighborhood of peer $v_i$. The function $n_{2l}(v_i)$ is defined accordingly and evaluates to the $2l$-neighborhood of peer $v_i$. A peer $v_i$ is referred to as a local maximum for item $j$ at a time $\tau$ if the worthiness of peer $v_i$ for item $j$ is greater than zero at time $\tau$ and peer $v_i$ has the smallest id among the peers with highest worthiness in $n_{2l}(v_i)$ at time $\tau$.

With these notations, we are able to formulate the distributed algorithm (see Algorithm 2). This algorithm requires only local knowledge of the network and computes a replica allocation for each data item in a distributed way. Moreover, the algorithm does not require explicit synchronization for replica assignment among the peers.

In the distributed algorithm, each peer periodically requests the worthiness of each peer in its $2l$-neighborhood for each data item (line 3). A simple method for a peer $v_i$ to obtain these worthiness values is to send a worthiness request message for each item $j$ to each peer in the $2l$-neighborhood of $v_i$. This message includes a timestamp with the current time $\tau$. Each peer that receives such a request message would then send back its worthiness for item $j$ at the time of the timestamp. The algorithm stops when the worthiness of each peer for each data item is zero (line 4) because all availability constraints are then satisfied. After a peer obtains the worthiness values for an item $j$ at a time $\tau$, the peer checks whether it was a local maximum for item $j$ at time $\tau$ (line 7) using the previously obtained worthiness values. Whenever a peer discovers that it was a local maximum for an item $j$, it assigns a replica of $j$ to itself (line 8). Because
Algorithm 2 Distributed algorithm

1: for each peer $v_i$ and each item $j$ in parallel do
2:  while true do
3:   let $\tau$ be the current time
4:   request worthiness of each peer in $n_{2l}(v_i)$ for item $j$ at time $\tau$
5:   if worthiness of peer $v_i$ for item $j$ at time $\tau$ equals 0 then
6:     stop computation at $v_i$ for $j$
7:   end if
8:   if peer $v_i$ was a local maximum for item $j$ at time $\tau$ then
9:     assign replica of item $j$ to peer $v_i$
10:    obtain replica of item $j$ from any replica holder
11:  end if
12: end while
13: end for

replicas of the same data item can be assigned in parallel, a peer is not necessarily the local maximum for an item $j$ anymore at the time it assigns a replica of $j$ to itself. (This is explained below in more detail.) However, we prove in the next section that a peer that is a local maximum for an item $j$ at some time will remain to be a local maximum for $j$ until this peer assigns a replica of $j$ to itself. Assigning a replica is independent from obtaining a replica (line 9). Hence, the computation of the final replica allocation is not delayed by obtaining replicas.

At each step, multiple peers can be local maxima. Hence, replica assignments can be performed in parallel as opposed to the centralized algorithm. The distributed algorithm may run with different speed on different peers and the time peers need to request worthiness values may vary for each peer and for each time the worthiness values are obtained. Thus, the order in which replicas are assigned to peers by the distributed algorithm is nondeterministic as opposed to the centralized algorithm.

Let $v_i$ be a peer that gets a replica assigned by the distributed algorithm and let $t_A(v_i)$ be the time of this replica assignment. In the same iteration where $v_i$ assigns a replica to itself (line 8), $v_i$ starts to obtain the worthiness values for all peers in its $2l$-neighborhood at an earlier time $t_O(v_i)$ (line 3). The delay between $t_O(v_i)$ and $t_A(v_i)$ results from obtaining the worthiness values of the peers in the $2l$-neighborhood of $v_i$ and from assigning a replica to $v_i$. There may be replica assignments to other peers during this delay and hence the replica allocation that was current at the time $t_O(v_i)$ may already be outdated at time $t_A(v_i)$. The nondeterministic order of replica assignments to peers and the delay between $t_A(p)$ and $t_O(p)$ are the main difficulties for proving in the next section that the distributed algorithm produces the same final replica allocation as the centralized algorithm.
While the algorithm is running, peers may go online and offline. The algorithm is guaranteed to terminate provided that peers only go temporarily offline but not permanently. This is because at any time and for each data item $j$ there is at least one peer that is a local maximum for item $j$. This peer will finally go online (if it was offline before), discover that it is a local maximum, and assign a replica of $j$ to itself.

### 4.4 Equivalence to the Centralized Algorithm

In this section, we prove that for any given input (i.e. any given set of query executions $Q$), the distributed algorithm computes the same replica allocation for each data item as the centralized algorithm. The equivalence of both algorithms implies that the approximation guarantee proved for the centralized algorithm (see Section 3.4) also holds for the distributed algorithm. Moreover, the results of the approximation evaluation of the centralized algorithm (see Section 3.6) also apply to the distributed one.

In this proof, we consider only a single data item since the DeterministicReplication problem (see Section 2.2) can be solved for each data item independently. In the following, data items are omitted when referring to allocations or the worthiness of peers. Hence, we use the notation $A$ for an allocation instead of $A_j$.

In this proof, we define a local maximum for an allocation $A \subseteq V$, not for a time $\tau$. This enables us to reason about all possible allocations, independent of whether these allocations occur during the distributed algorithm or not. More precisely, we define a peer $v_i$ to be a local maximum for an allocation $A \subseteq V$ if peer $v_i$ has the smallest id among all peers with the highest worthiness at allocation $A$. Let $w: V \times \mathcal{P}(V) \rightarrow \mathbb{N}$, such that $w(v_i, A)$ evaluates to the worthiness of peer $v_i$ at allocation $A$. Let $lmax : \mathcal{P}(V) \rightarrow \mathcal{P}(V)$, such that $lmax(A)$ evaluates to the set of peers that are local maxima at allocation $A$. The global maximum of an allocation $A$ is defined as the peer that has the smallest id among all peers in $V$ with highest worthiness at allocation $A$.

The centralized algorithm (see Section 3.3) can easily be generalized to start with any given initial allocation $A$ by assigning a corresponding initial set of indices to $J^*$ in line 1 of Algorithm 1. Then the centralized algorithm deterministically produces an output allocation for any given initial allocation $A$ and any given set of queries $Q$. Let $f : \mathcal{P}(V) \rightarrow \mathcal{P}(V)$, such that $f(A)$ evaluates to the final replica allocation produced by the centralized algorithm starting with allocation $A$. Note that $A \subseteq f(A)$ for any allocation $A \subseteq V$ and that $f(A) = A$ if all availability constraints are satisfied at allocation $A$. 
Lemma 3. Let \( v \in V, A \subseteq V \) and let \((v_1, \ldots, v_d)\) be the sequence of replica assignments produced by the centralized algorithm starting with allocation \( A \). Let \( A_i = A \cup \{v_1, \ldots, v_i\} \) be the replica allocation obtained after \( i \) replica assignments of the centralized algorithm for all \( i = 0, \ldots, d \). If \( v \in \text{lmax}(A) \) then there exists an \( s \in \{1, \ldots, d\} \) such that \( v = v_s \) and we have \( v_s \in \text{lmax}(A_i) \) for all \( i = 1, \ldots, s - 1 \).

**Proof.** Assume \( v \in \text{lmax}(A_0) \). Then \( w(v, A_0) > 0 \) and since the worthiness of each peer that has a replica assigned equals zero, we have \( v \notin A_0 \). Let \( s' \in \{1, \ldots, d\} \) be the largest index \( i \) for which \( v \notin A_i \). Since \( v \notin A_0 \), we have \( s' \geq 0 \). We first show by induction that

\[
v \in \text{lmax}(A_i) \text{ for all } i = 0, \ldots, s'.
\]  

(Note that Equation (4.1) does not imply that the centralized algorithm assigns a replica to \( v \).) For \( i = 0 \), we have \( v \in \text{lmax}(A_0) \) by assumption. Under the assumption that \( v \in \text{lmax}(A_\kappa) \) for any \( \kappa \in \{1, \ldots, s' - 1\} \), we have to show that \( v \in \text{lmax}(A_{\kappa+1}) \). Since \( \kappa + 1 \leq s' \), we have \( v_{\kappa+1} \neq v \). Since \( v \in \text{lmax}(A_\kappa) \) by induction assumption, none of the peers in \( n_2(v)\setminus\{v\} \) can be the global maximum of \( A_\kappa \) and hence \( v_{\kappa+1} \in V\setminus n_2(v) \). Only the query executions of peers in \( n_2(v) \) can sample \( v \) and the query executions of peers in \( n_2(v) \) cannot sample peers in \( V\setminus n_2(v) \). Hence, assigning the next replica to any peer in \( V\setminus n_2(v) \) cannot affect the worthiness of \( v \). Thus, \( w(v, A_\kappa) = w(v, A \cup \{v_{\kappa+1}\}) = w(v, A_{\kappa+1}) \). Since the worthiness of \( v \) is the same at \( A_\kappa \) and \( A_{\kappa+1} \), and the worthiness of all other peers in \( n_2(v) \) may only decrease from \( A_\kappa \) to \( A_{\kappa+1} \), \( v \) is still a local maximum at \( A_{\kappa+1} \), i.e. \( v \in \text{lmax}(A_{\kappa+1}) \).

We now prove that \( v \) gets a replica assigned by the centralized algorithm, that is, \( v \in f(A) \). Since \( v \in \text{lmax}(A) \) implies \( w(v, A) > 0 \), there is at least one peer \( v' \in n_2(v) \) whose availability constraint is not satisfied at allocation \( A \). The centralized algorithm must therefore assign a replica to at least one peer in \( n_2(v) \) in order to satisfy the availability constraints of \( v' \). By Equation (4.1), of all the peers in \( n_2(v) \), the centralized algorithm must first assign a replica to \( v \) and hence \( v = v_s \) for one \( s \in \{1, \ldots, d\} \). Then \( s' = s - 1 \) and by Equation (4.1) we have \( v_s \in \text{lmax}(A \cup \{v_1, \ldots, v_i\}) \) for \( i = 1, \ldots, s - 1 \). \( \square \)

Lemma 4. Let \( v \in V \) and \( A \subseteq V \). Let \((v_1, \ldots, v_d)\) be the sequence of replica assignments produced by the centralized algorithm starting with allocation \( A \) and let \( A_i = A \cup \{v_1, \ldots, v_i\} \) for \( i = 0, \ldots, d \). If \( v \in \text{lmax}(A) \) then there exists an \( s \in \{1, \ldots, d\} \) such that \( v = v_s \) and we have \( w(v_i, A \cup v_s) = w(v_i, A) \) for all \( i = 1, \ldots, s - 1 \).

**Proof.** Assume \( v \in \text{lmax}(A) \). By Lemma 3, there exists an \( s \in \{1, \ldots, d\} \) such that \( v = v_s \). We first observe that \( v_i \notin n_2(v_s) \) for \( i = 1, \ldots, s - 1 \). We have
$v_i \in \text{Imax}(A_{i-1})$ for $i = 1, \ldots, s - 1$ by assumption (for $i = 1$) and by Lemma 3 (for $i = 2, \ldots, s - 1$). Hence, if $v_i$ was in $n_2(v_s)$ for any $i \in \{1, \ldots, s - 1\}$ then $v_i$ could not be the global maximum at allocation $A_{i-1}$.

Since assigning a replica to $v_s$ can only affect the worthiness of peers in $n_2(v_s)$ and since $v_i \notin n_2(v_s)$ for $i = 1, \ldots, s - 1$, we have $w(v_i, A \cup \{v_s\}) = w(v_i, A)$ for $i = 1, \ldots, s - 1$. \hfill \Box

**Lemma 5.** Let $v \in V$ and let $A \subseteq V$. If $v \in \text{Imax}(A)$ then $f(A \cup \{v\}) = f(A)$.

**Proof.** Let $(v_1, \ldots, v_d)$ be the sequence of replica assignments produced by the centralized algorithm starting with allocation $A$ and let $A_i = A \cup \{v_1, \ldots, v_i\}$ for $i = 0, \ldots, d$. By Lemma 3, there exists an $s \in \{1, \ldots, d\}$ such that $v = v_s$.

Let $f_i(A)$ be the allocation that results after $i$ replica placements of the centralized algorithm starting with allocation $A$ and let $g_i(A) = f_i(A) \setminus A$. We first show by induction that $g_i(A \cup \{v_s\}) = g_i(A)$ for $i = 0, \ldots, s - 1$. For $i = 0$ we have $g_0(A \cup \{v_s\}) = g_0(A) = \emptyset$. Under the assumption that $g_\kappa(A \cup \{v_s\}) = g_\kappa(A)$ for any $\kappa \in \{0, \ldots, s - 2\}$, we have to show that $g_{\kappa+1}(A \cup \{v_s\}) = g_{\kappa+1}(A)$. We have that $v_{\kappa+1}$ is the global maximum of allocation $A_\kappa$. By Lemma 3, $v_s \in \text{Imax}(A_\kappa)$ and hence $w(v_{\kappa+1}, A_\kappa \cup \{v_s\}) = w(v_{\kappa+1}, A_\kappa)$ by Lemma 4. Moreover, assigning a replica to $v_s$ at allocation $A_\kappa$ may only decrease the worthiness of all other peers. Hence, $v_{\kappa+1}$ is also the global maximum at allocation $A_{\kappa+1} \cup \{v_s\}$ and thus

$$g_{\kappa+1}(A \cup \{v_s\}) = g_\kappa(A \cup \{v_s\}) \cup \{v_{\kappa+1}\}$$

$$= g_\kappa(A) \cup \{v_{\kappa+1}\} \quad \text{(by induction assumption)}$$

$$= g_{\kappa+1}(A).$$

For $i = s - 1$, we obtain $g_{s-1}(A \cup \{v_s\}) = g_{s-1}(A)$ and hence $f_{s-1}(A \cup \{v_s\}) = (A \cup \{v_s\}) \cup g_{s-1}(A \cup \{v_s\}) = A \cup g_{s-1}(A) \cup \{v_s\} = f_s(A)$. The sequence of further replica assignments produced by the centralized algorithm is then identical after $f_s(A)$ and $f_{s-1}(A \cup \{v_s\})$ and hence $f(A \cup \{v\}) = f(A \cup \{v_s\}) = f(A)$. \hfill \Box

We now consider the behavior of the distributed algorithm from a centralized perspective. Without loss of generality, we assume that only one replica is placed at a time by the distributed algorithm. Let $(\vartheta_1, \ldots, \vartheta_d')$ be the sequence of replica assignments produced by the distributed algorithm starting with allocation $B_0$ and let $B_i = B_0 \cup \{\vartheta_1, \ldots, \vartheta_i\}$ be the replica allocation obtained after $i$ replica assignments for all $i = 1, \ldots, d'$.

Moreover, we use the notion of a pool of candidates for replica placement. When the distributed algorithm starts, the pool is empty. Each peer $v$ that gets a replica assigned by the distributed algorithm is added at time $t_O(v)$ to the pool
and is removed at time $t_A(v)$ from it. Let $P_i \subseteq V$ denote the pool of replica placement candidates just before the $i$th replica assignment. Then $\vartheta_i \in P_i$ for all $i = 1, \ldots, d'$. Between the $i-1$th and the $i$th replica assignment, a possibly empty set $T_i$ of peers is added to the pool. Since the replica allocation does not change between the $i-1$th and the $i$th replica assignment, we have $T_i \subseteq \text{lmax}(B_{i-1})$ for all $i = 1, \ldots, d'$. Then $P_1 = T_1$ and $P_{i+1} = (P_i \setminus \{v_i\}) \cup T_{i+1}$ for all $i = 1, \ldots, d' - 1$.

**Lemma 6.** Let $(\vartheta_1, \ldots, \vartheta_d)$ be a sequence of replica assignments produced by the distributed algorithm starting with allocation $B_0 \subseteq V$ and let $B_i = B_0 \cup \{\vartheta_1, \ldots, \vartheta_i\}$ for $i = 1, \ldots, d'$. Then $\vartheta_i \in \text{lmax}(B_{i-1})$ for all $i = 1, \ldots, d'$.

**Proof.** We first show by induction that $P_i \subseteq \text{lmax}(B_{i-1})$ for $i = 1, \ldots, d'$. For $i = 1$ we have $P_1 = T_1 \subseteq \text{lmax}(B_0)$. Under the assumption that $P_\kappa \subseteq \text{lmax}(B_{\kappa-1})$ for any $\kappa \in \{1, \ldots, d' - 1\}$, we have to show that $P_{\kappa+1} \subseteq \text{lmax}(B_\kappa)$. Since $P_\kappa \subseteq \text{lmax}(B_{\kappa-1})$, no peer in $P_\kappa$ is in the $2l$ neighborhood of another peers in $P_\kappa$. Hence, assigning the next replica to $\vartheta_\kappa \in P_\kappa$ at allocation $B_{\kappa-1}$ cannot affect the worthiness of peers in $P_\kappa \setminus \{\vartheta_\kappa\}$, i.e. $w(v, B_{\kappa-1}) = w(v, B_{\kappa-1} \cup \{\vartheta_\kappa\}) = w(v, B_\kappa)$ for each peer $v \in P_\kappa \setminus \{\vartheta_\kappa\}$. Moreover, the worthiness of all peers not in $P_\kappa \setminus \{\vartheta_\kappa\}$ may only decrease from $B_{\kappa-1}$ to $B_\kappa$. Hence, $P_\kappa \setminus \{\vartheta_\kappa\} \subseteq \text{lmax}(B_\kappa)$ and since $T_{\kappa+1} \subseteq \text{lmax}(B_\kappa)$, we have $P_{\kappa+1} = (P_\kappa \setminus \{\vartheta_\kappa\}) \cup T_{\kappa+1} \subseteq \text{lmax}(B_\kappa)$.

Since $\vartheta_i \in P_i \subseteq \text{lmax}(B_{i-1})$ for all $i = 1, \ldots, d'$, we have $\vartheta_i \in \text{lmax}(B_{i-1})$ for all $i = 1, \ldots, d'$.

**Theorem 3.** For any given initial replica allocation, the distributed algorithm produces the same final replica allocation as the centralized algorithm.

**Proof.** Let $(\vartheta_1, \ldots, \vartheta_{d'})$ be a sequence of replica assignments produced by the distributed algorithm starting with allocation $B_0$ and let $B_i = B_0 \cup \{\vartheta_1, \ldots, \vartheta_i\}$ for $i = 1, \ldots, d'$. To prove the theorem, we have to show that $B_{d'} = f(B_0)$.

At allocation $B_0$, the distributed algorithm assigns the next replica to $\vartheta_1$. Since $\vartheta_1 \in \text{lmax}(B_0)$ by Lemma 6, we have $f(B_1) = f(B_0 \cup \{\vartheta_1\}) = f(B_0)$ by Lemma 5. Applying Lemma 6 and 5 iteratively for each replica assignment of the distributed algorithm, we obtain $f(B_0) = f(B_1) = \cdots = f(B_{d'})$. Since all availability constraints are satisfied after $d'$ replica placements of the distributed algorithm, we have $f(B_{d'}) = B_{d'}$ and hence $B_{d'} = f(B_{d'}) = f(B_0)$. (Note that this implies $d' = d$ in case $A_0 = B_0$.)

In our case, the initial allocation is the empty set, also if replicas have been placed in the network. However, non-empty initial allocations could be used to find near-optimal replica allocations if some data items may not be removed by the replication algorithm.
4.5 Efficiency of the Distributed Algorithm

An important characteristic of the efficiency of the distributed algorithm is its time required to construct replica allocations. As a measure for this construction time, we use the number of rounds required by the distributed algorithm for converging to a stable allocation. The number of rounds is defined by the following execution model of the distributed algorithm. At each round, each peer computes whether it is a local maximum or not. Each peer that is a local maximum gets a replica assigned before the next round starts. This procedure is repeated until all availability constraints are satisfied.

The number of rounds required by the distributed algorithm is constrained by the graph topology and the length of random walks. This is because any two peers in the network can only be local maxima for a particular data item at the same time if the shortest path between both peers has a length of more than $2l$. (Recall that a peer $v_i$ is a local maximum for an item $j$ at a time $\tau$ if the worthiness of peer $v_i$ for item $j$ is greater than zero at time $\tau$ and peer $v_i$ has the smallest id among the peers with highest worthiness in $n_{2l}(v_i)$ at time $\tau$.) However, the $2l$-neighborhoods of peers that are local maxima may overlap. If the graph has a diameter of at most double walk length, only a single peer can be a local maximum at a time and in each round, only one replica is assigned to a peer.

In graphs where the diameter is larger than double walk length, the distributed algorithm may finish in few rounds. For instance, the distributed algorithm finishes already after one round if the walk length is zero or if there are no edges in the graph because the $2l$-neighborhood of each peer then only contains the peer itself. Another, more realistic, example is that only few peers issue queries in the network and none of these query initiators is in the $2l$-neighborhood of another query initiator. In this case, random walks of query executions take various paths in the $l$-neighborhood of each query initiator and hence each query initiator is likely to have the highest worthiness in its $2l$-neighborhood. Then, the distributed algorithm assigns a replica to each query initiator and finishes after a single round.

In graphs with a large diameter, it can also happen that only a single replica can be assigned in each round. An example of such a situation is shown in Figure 4.2a. In this example, we have a network chain topology with six peers. Moreover, we have a set of query executions (shown as arrows) for the same data item with a random walk length of one. Note that this example can easily be extended to any graph diameter and to larger walk lengths. Each peer has an availability constraint of 100%, so the worthiness of each peer (shown as numbers) is the number of unsatisfied query executions that sample this peer. For the set of
query executions given in this example, the worthiness of peers increases from left to right, except for the last peer. Since each peer has its right neighbor in its 2l-neighborhood and the rightmost peer has its left neighbor in its 2l-neighborhood, only the peer with a worthiness of five is a local maximum in this situation. In the first round, the distributed algorithm therefore assigns a replica to this peer.

Figure 4.2b shows the situation after this replica assignment. The peer that is assigned a replica is shown with a white fill, successful query executions are removed, and the worthiness of peers is updated. Again, only one peer (the one with a worthiness of three) is a local maximum. Figure 4.2c shows the situation after the distributed algorithm assigns a replica to this peer. Then both the first and the second peer (from left) have a worthiness of one. If we assume that the first peer has a smaller id than the second peer does, then the distributed algorithm assigns a replica to the first peer. Finally, Figure 4.2d depicts the final replica allocation of the distributed algorithm. This example shows that a sparse graph topology with a large diameter is only a necessary but not a sufficient condition for a high efficiency of the distributed algorithm.

Mobile P2P networks are a prime example for network topologies with a large graph diameter. In mobile P2P networks, each peer is connected to the peers that are within a certain communication radius. Because of this characteristic, mobile P2P networks seem to be a suitable application field for our algorithm.
In this section, we analyzed the efficiency of the distributed algorithm for special cases of P2P topologies and query executions. In the next section, we experimentally evaluate the efficiency of the distributed algorithm for P2P settings.

4.6 Experiments on the Efficiency of the Distributed Algorithm

In this section, we present an experimental evaluation of the distributed algorithm proposed in Section 4.3. The goal of this evaluation is to investigate the efficiency of the distributed algorithm in terms of the time required to construct replica allocations in P2P settings. To this end, we performed several simulations with random graphs. In Section 4.6.1, the simulation setup and the implementation of the simulation are described. In Section 4.6.2, the simulation results are presented and discussed.

4.6.1 Simulation Setup and Implementation

We investigate the efficiency of the distributed algorithm for graphs where the diameter is much larger than random walk length since only then it is possible to place many replicas per round (see Section 4.5). We construct sparse graphs with a large diameter by generating random graphs with a small edge probability (probability for each possible edge to occur in the random graph).

However, for small edge probabilities, the graph is likely to be separated into many components. This distorts the number of rounds needed by the distributed algorithm because in each component at least one replica can be placed per round. For instance, an edge probability of zero results in a graph without any edges and then only a single round is needed by the distributed algorithm. Hence, we aim to generate connected graphs with a large diameter. To this end, we generate random graphs with a small edge probability that is still large enough to obtain a component with more than half of all vertices. The generated graph is then pruned by discarding the vertices that do not belong to this component.

For the simulation of searching in the network, we use the same stochastic model as we used for the evaluation of the approximation quality (see Section 3.6), that is, we have uniform query rates and exactly one query is executed at each step. After issuing a certain number of queries in the network, we use the distributed algorithm to compute a replica allocation that satisfies the availability constraints and we count the number of rounds needed to obtain this allocation. In order to determine which parameters affect the efficiency of the distributed algorithm, we vary each of the following parameters separately:
Table 4.1: Default parameter values for evaluating the algorithm efficiency

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of peers (before graph pruning)</td>
<td>500</td>
</tr>
<tr>
<td>Edge probability</td>
<td>0.003</td>
</tr>
<tr>
<td>Number of queries issued in the network</td>
<td>10000</td>
</tr>
<tr>
<td>Random walk length</td>
<td>3</td>
</tr>
<tr>
<td>Availability constraint of each peer</td>
<td>0.8</td>
</tr>
<tr>
<td>Online probability of each peer</td>
<td>0.8</td>
</tr>
</tbody>
</table>

- Edge probability
- Number of queries issued in the network
- Random walk length
- Availability constraint of each peer
- Online probability of each peer

Parameters not varied during a simulation run are assigned a default value (see Table 4.1). Moreover, we perform 100 simulation runs for each parameter configuration. At each run, a new random graph and a new set of queries is generated. In case a generated graph does not contain a component with more than half of all vertices, we repeat graph generation until such a component is obtained. As with evaluating the approximation quality of the centralized algorithm (see Section 3.6), it would also be interesting in these experiments to use heterogeneous query rates, online probabilities, and availability constraints.

For 100 simulation runs with the default edge probability of 0.003, the number of peers in the largest component is 295.87 on average with a sample variance (see Wasserman [Was04], for instance) of 622.20 and the graph diameter is 28.33 on average with a sample variance of 23.15.

### 4.6.2 Results and Discussion

Figures 4.3-4.9 show how the different parameters affect the number of rounds (defined in Section 4.5) needed by the distributed algorithm. We also included the number of replicas in the figures since this quantity is an upper bound for the number of rounds. The graphs show averages over 100 runs. Over all sets of 100 runs, the sample variance of the number of rounds is at most 312.71 and the sample variance of the number of replicas is at most 734.77.
Our main observation is that the efficiency of the distributed algorithm is mainly determined by the proportion between walk length and graph diameter. In case the graph diameter is much larger than random walk length, the distributed algorithm requires only few rounds to construct replica allocations.

Figure 4.3 shows the number of rounds and the number of replicas as the edge probability increases. The graph starts with an edge probability of 0.003 since the probability to obtain a component with more than half of all vertices is not large enough for smaller edge probabilities. For this edge probability, we can observe that a small average number of rounds (40.65) is needed compared to the average number of replicas (158.59). This indicates that the distributed algorithm efficiently assigns replicas to peers in P2P topologies with a large graph diameter.

When the edge probability increases, the number of rounds quickly increases. This is because the graph diameter quickly decreases with an increasing edge
probability (see Figure 4.4). Recall that any two peers can only be local maxima for the same data item at the same time if they are not connected by a path with a length of at most $2l$ (see Section 4.5). Increasing the edge probability hence increases the probability that such a path exists. Moreover, we observe that the number of replicas increases for up to an edge probability of 0.0050. This is because the number of peers in the largest component increases with an increasing edge probability. The increasing number of replicas is therefore also a cause for the increasing number of rounds.

Figure 4.5 shows the number of rounds and the number of replicas as the length of random walks increases. For a walk length of zero, only a single round is required because the $2l$-neighborhood of each peer contains only the peer itself. When the walk length increases, the maximum number of local maxima at any given round decreases and hence the number of rounds increases. The number of rounds finally converges to the number of replicas since for large walk lengths only one replica can be placed in each round.

Figures 4.6-4.9 show the results as the number of issued queries, the availability constraints of each peer, and the online probabilities of each peer increases. Since the number of rounds is always in proportion to the number of replicas, these parameters seem to influence the number of rounds only indirectly by the number of replicas. Overall, the number of rounds is only weakly affected by these parameters. Moreover, the trends for the number of replicas are very similar to the evaluation results of the centralized algorithm (see Section 3.6) and are therefore not discussed here again.

For varying the online probability of each peer (Figure 4.9), one might wonder why the number of rounds is not one for very small online probabilities. In this case at most one peer is online at a time and hence each query execution only
CHAPTER 4. DISTRIBUTED ALGORITHM

Figure 4.6: Efficiency results for up to 1000 queries

Figure 4.7: Efficiency results for up to 20000 queries

Figure 4.8: Efficiency results for different availability constraints
samples the query initiator, which might appear similar to an edge probability of zero or a random walk length of zero. However, in contrast to edge probabilities and random walk lengths, varying online probabilities does not affect the $2l$-neighborhoods of peers.

Figure 4.9: Efficiency results for different online probabilities
In this thesis, we presented a distributed approximation algorithm for replication in unstructured P2P networks together with a proof of its approximation guarantee. The goal of the distributed algorithm is to find a replica assignment to peers that satisfies given availability constraints and minimizes the number of replicas for each data item. In contrast to prior replication approaches in unstructured P2P networks, our approach aims to consider individual peer requirements regarding search success in the presence of temporary peer outages.

To devise the distributed algorithm, we first studied the problem in a centralized setting (i.e. with complete knowledge of the network) and developed a centralized algorithm. We then turned the centralized algorithm into a distributed one by exploiting local properties of the centralized algorithm. We proved the approximation guarantee of the distributed algorithm by first proving the same approximation guarantee for the centralized algorithm and then showing that for any input (i.e. any given set of query executions), the distributed algorithm produces the same replica assignment to peers as the centralized one.

Our experimental evaluation on random graphs showed that the approximation solution of the centralized algorithm and hence also the approximation solution of the distributed algorithm is close to the optimum solution in terms of required number of replicas. Moreover, we experimentally evaluated the efficiency of the distributed algorithm in terms of the time required to construct replica allocations. We found that the algorithm efficiently performs replica assignments to peers for sparse network graphs with a large diameter. This makes the distributed algorithm particular suitable for mobile P2P environments.
As a byproduct of our investigation, we developed an approximation algorithm for a generalization of the partial set cover problem and proved an approximation guarantee for this algorithm. In this generalized problem, we have multiple disjoint ground sets instead of a single ground set. For each ground set, a fraction of elements to be covered can be specified separately.

Future work could be directed towards efficient protocols for the distributed algorithm. In Chapter 4, we stated that a peer can send a request to all the peers in its $2l$-neighborhood to determine its own worthiness or to check whether it is a local maximum for a data item. However, these actions have to be performed periodically, which clearly incurs a multitude of network messages. Another open issue of the distributed algorithm is that peers that are local maxima of an item $j$ need to know from which other peers they can obtain a replica of $j$. For this purpose, a gossiping protocol could be employed to spread this information. The number of messages required for such a gossiping protocol could be reduced by attaching information to query messages.

In this thesis, we focused on computing replica assignments to peers for steady states of the network. However, real networks do not stay in a steady state and hence replica assignments need to be recomputed after some time. It seems desirable to guide the frequency of re-computations by the stability of network parameters, such as query rates and online/offline behavior of peers. In this context, it is interesting to investigate whether small changes in network parameters could lead to significant changes in the replica assignment to peers computed by the distributed algorithm. Such a behavior would have the disadvantage that moving replicas at the transition from one replica allocation to the next would cause considerable network traffic. In this case, it is then interesting to explore methods for “smoothing” the transition from one replica allocation to the next.

Other future work includes proving our conjecture about the NP-hardness of the deterministic replication problem (DeterministicReplication). Future work also includes evaluating the approximation quality and the efficiency of the distributed algorithm with heterogeneous parameters (query rates, online probabilities, and availability constraints) and with data of real P2P networks. Moreover, it would be interesting to generalize our approach to other search strategies such as issuing multiple random walks per query execution, so-called walkers [LCC+02]. This search method achieves faster query responses compared to an increased maximum length of random walks [LCC+02]. Moreover, multiple walkers have the advantage that they do not affect the $2l$-neighborhoods of peers and hence they might not affect the efficiency of the distributed algorithm.

In our approach, we observe the network for some time in order to obtain a set of query executions $Q$ as the input of our algorithm. An alternative approach is to
estimate network parameters such as query rates and online/offline behavior of peers by considering user preferences, for instance. This has the advantage that replica allocations could be computed and performed before peers start issuing queries and approximation constraints could then be satisfied with the first query being issued. However, in this approach a set of query executions $Q$ needs to be computed from network parameters, which is not be straightforward to perform in a distributed way.

Finally, long-term directions of our future work are to consider further constraints of real P2P networks such as storage and load capacities of peers. A first approach to consider such capacities is to assign a cost to each set $S_j$ in the WeightedMulti-PartialSetCover problem (see Section 3.3), such that the assigned cost is inversely proportional to the capacity of the peer corresponding to $S_j$. 
Bibliography


