Fast Construction of Efficient Structured Peer-to-Peer Overlays

Lokman Rahmani

INRIA Sophia Antipolis - Méditerranée
2004 Route des Lucioles
06560 Sophia Antipolis, France

Stage de Fin d'Etudes
2013

[R]

Fast Construction of Efficient Structured Peer-to-Peer Overlays

Student: Lokman RAHMANI
lokmanet@gmail.com - Master 2 IFI - Spécialité CSSR
Supervisor : Fabrice HUET
fabrice.huet@inria.fr - INRIA - Équipe OASIS
Academic tutor : Arnaud LEGOUT

Internship total budget : 875h

Contents

ABSTRACT

This report presents algorithms for building *Distributed Hash Tables (DHT)* or *Structured Overlays*. DHTs are used as self-managing system to deal with distributed data over large networks.

The provided algorithms **guarantee]** building Efficient DHTs by ensuring selected properties, and this for the general case. Our algorithms can be mainly used in DHT-based storage systems to build efficient systems or to recover the last configuration of the systems after crash very fast.

In the general case, the managed data distribution can be non-uniform; and if we build the DHT using canonical algorithms this will create load imbalance over/among participating machines/peers: it will take a lot of time and will lead to a DHT with poor performances.

We will first define a general, easy-to-use model that can be used to consider several properties we want to have to ensure efficiency/performances. Based on that formal model, we provide a distributed structure that permit to reorganize the DHT to facilitate the join of new machines/peers while considering the selected properties. This structure was used to ensure both load- and traffic-balancing over DHT machines/peers.

DHTs are known as distributed structures: each participating peer maintains a local state of the DHT which includes itself and a small number of its neighbors; no peer have a global view of the whole DHT. Actually this is an important **obstacle]** to fast build DHT, when considering some properties related to new peers characteristics. Previous solutions to ensure such properties are mostly not distributed and/or problem-specific.

We provide an algorithm that build a partial view of the DHT more interesting than the local one. After based on this partial view, we develop a heuristic that minimize data transfers by considering the input data of each new peer. This reduces clearly the time to build an operational DHT.

All the algorithms we will present have been implemented and tested using simulation. We will talk briefly about the simulation tools and environment. We will also present and discuss our experimental results.

ACKNOWLEDGEMENTS

1

Introduction

Nowadays applications generate, consume and process a large amount of data. This make the need to provide platforms and systems that can deal with this huge load of data.
Distributed hash tables (DHTs) are very used to manage distributed data in an efficient manner. In a network with many machines, Data objects can be stored and retrieved in short time from any machine of the network.
The aim of this chapter is to introduce the topic. First we will talk about DHTs, their advantages, and some of their applications. After we will present our contributions on building efficient DHTs and compare them to the related work. Finally the organization of the report is presented.

1.1 What is a DHT?[?]

A distributed hash table is, as its name suggests, a hash table which is distributed among a set of cooperating computers, which we refer to as peers. Just like a hash table, it contains key/value pair. The main service provided by a DHT is the lookup operation, which returns the value associated with any given key. In the typical usage scenario, a client has a key for which it wishes to find the associated value. Thereby, the client provides the key to *any one* of the nodes, which then performs the lookup operation and returns the value associated with the provided key. Similarly, a DHT also has operations for managing items, such as inserting and deleting data items.
The representation of the key/value pairs can be arbitrary. For example, the key can be a string or an object. Similarly, the value can be a string, a number, or some binary representation of an arbitrary object. The actual representation will depend on the particular application.
An important property of DHTs is that they can efficiently handle large amounts of data items. Because of limited storage/memory capacity and the cost of inserting and updating items, it is infeasible for each node to locally store every item. Therefore, each node is *responsible* for part of the data items, which it stores locally.
As we mentioned, every node should be able to lookup the value associated with any key. Since all items are not stored at every node, requests are routed whenever a node receives

a request that it is not responsible for. For this purpose, each node has a *routing table* that contains pointers to other nodes, known as the node's *neighbors*. Hence, a query is routed through the neighbors such that it eventually reaches the node responsible for the provided key.

Consistent Hashing]
A hash table uses a hash function applied to keys to get an index where to store associated data items. Just like hash table, the DHT also uses a hash function to decide in which peer a key/value item will be stored. The difference is that the hash function maps keys to a pre-defined *virtual space* (e.g. 2-Dimensional Cartesian space) which is more general than integer indexes. An addition is that peers will have identifiers from the same virtual space. Then, each peer will be responsible on keys/values items that are close to him considering some defined metric. This is what is called *consistent hashing*. Consistent hashing was introduced in the context of web pages caching on multiple node in the Internet[**?**]. To ensure load balance over peers, the hash function is supposed to be *uniform*: the stored keys/values items are uniformly distributed over the virtual space; such that each peer will store approximatively the same amount of data items.

Range Queries In some applications, it might be useful to ask the DHT to find values associated to all keys in a numerical or an alphabetical range. For example, in the context of the Play[**?**] European project, the keys represent XXXX. Hence an application might query the DHT to search for all key in the interval XXXXX. DHTs doesn't support naturally ranges queries. Over the last years, many works have been done to consider range queries [**?**]. The Play project used the solution proposed by XXX and al. in [**?**]: the use of *order-preserving* hash function to map keys to the virtual space. Formally an order-preserving hash function h_{op} is a function that ensure if we have two keys k_1, k_2 and $k_1 < k_2$ then we have $h_{op}(k_1) <_v h_{op}(k2)$, where $<$ and $<_v$ are ordering relation defined in the keys space and virtual space respectively. From this definition a uniform hash function is clearly not an order-preserving hash function. So When using such hash functions this will lead to load-imbalance over peers. **-Very important this last sentence-]** In this report we present algorithms that ensure load-balance and can ensure multiple criteria even when using order-preserving or non-uniform hash function in general.

Building DHTs
DHTs are built incrementally. The first peer will manage all DHT data. As well as new peer arrives and join the DHT, it becomes responsible of a part of DHT data that it get from a unique DHT peer. This peer is called the *join peer*.
For a new peer to know about the DHT and select the join peer, most of DHTs make use of a directory that [will] maintains a list of peers that are participating to the DHT. So a new peer will first contact the directory to get a random DHT peer, called *entry peer*. The new peer calculate then its identifier (based on information about the DHT that it get from the directory or the entry peer) and ask the entry peer to find the DHT peer with identifer that is closest to its identifier. Once found, the insertion is done inducing the new peer, the join peer and its neighbors.

Structured Overlay networks
DHT is said to construct an *overlay network* because its peers are connected to each other by links established at the application layer and are completely different from physical links that connect machines that are hosting peers. Figure **??** ilustrate an overlay network and the physical network [that it's built on top of it]/[it relays on]. DHTs are classified as *structured* overlay networks as the links forming the overlay are established between peers in a way to form a specific topology (a ring, tree, ...).

Structured Overlay networks[?] - Option 2
A DHT is said to construct an *overlay network*, because its nodes are connected to each other over an existing network, such as the Ethernet ot Internet, which the overlay uses to provide its own routing functionality. The existing network is then referred to as the *underlay network*. If the underlay network is the Internet, the overlay routes requests between the nodes of the DHT, and each such reroute passes through the routers and switches which form the underlay. DHTs are classified as *structured* overlay networks as the links forming the overlay are established between peers in a way to form a specific topology (a ring, tree, ...).
Figure **??** illustrates an overlay network and its corresponding underlay network.

Par abus de language,] we will omit the word 'structured' in structured overlays.

1.2 Applications on top of DHTs

We have now described what is a DHT and ...], in this section we will talk very briefly about some applications that use DHTs. In almost all cases, DHTs are not used comme telles], but serves as a lower layer to manage data of distributed system as we will see.

Publish/Subscribe systems In a Publish/Subscribe system two types of roles] exist; as its name suggests : publisher and subscriber. Publishers are those who generate and provide contents. Inversement] Subscribers are content consumers. A subscriber can present its interest to a part of content by making subscriptions to the topics he want to receive content about. The role of a Publish/Subscribe system is to save all subscriptions, and when receiving content on a topic, forward it to all subscriptions for that topic.
Its clear that with the presence of many publisher and many subscribers that the system will have a huge load of content to receive, store and deliver in/at real-time. This become more difficult when publishers and subscribers are geographically distributed]. This is the aim of the Play project to build a Publish/Subscribe system that covers most of Europe. OASIS team; as a responsible of data management part; adapted a DHT-based storage system [?].– Many other works have been done on using DHTs in Publish/Subscribe systems [?]–]. This work is in that context to build efficient DHT-Based storage considering several properties.

Cloud Key/Value Storage Very recent usage of DHT-Based storage is their adaptation in the context of Cloud Computing. In the Cloud model/philosophy, companies (like Google, Amazon) provide solutions and applications for their clients transparently and can be used virtually from any where using very poor devices with a very high availability. where all computation and data management is done in the server side, the client side have to provide resources only for the visualization.
To ensure that, big companies deploy all over the world many powerful data-centers. In this case the big deal is to ensure high-availability where the systems receive accesses and queries from thousands of clients from all the world. To meet this requirements, many DHT-based storage systems have been proposed and deployed, augmented by some additional properties and are known as key/value storage [**?**]. **Our work can be also used well in that context.]**

Other uses of DHTs
DHTs have been used in many other contexts. we can cite :

- Host discovery and mobility: like Internet Indirection Infrastructure (i3) [**?**], Host Identity Payload (HIP)[**?**], ...
- Web caching and web servers: Squirrel[**?**], DKS Organized Hosting (DOH)[**?**]
- In peer-to-peer file sharing applications : BitTorrent[**?**], Azureus, eMule, and eDonkey use the Kademlia DHT[**?**].

1.3 Contributions

Our work is principally about building DHTs. This section resumes our contributions and compare them to related work. Later, each contribution is described more in details, and main results are presented and discussed; each contribution in separate chapter.

1.3.1 Model for Building Efficient DHTs

To build rapidly efficient DHTs we need to ensure several properties at the same time. Based on a general definition of DHTs, we provide **a model to build efficient DHTs in reasonable] time ensuring selected properties]**. We modeled this problem as an ILP[1] problem. Each property to ensure is called *criterion*. Each criterion is associated a cost. All selected criteria are **grouped]** in an *objective function* that we want to optimize (minimize or maximize).
When a new peer want to join the DHT, it have to find the best peer to be inserted by it so that it optimizes the objective function and then satisfy most of the criteria. In this model we distinguish two types of criteria intra-DHT criteria and extra-DHT criteria. Intra-DHT criteria **are criteria that]** can be calculated based only on the DHT state like load-balancing and traffic-balancing, and thus are independent from the new peer that want to join the

[1]Integer Linear Programming

DHT. In the other side an extra-DHT criteria depends also on the new peer and their evaluation change for each new peer. An example of an extra-DHT criteria is data transfer minimization where the quantity of data transfers will depends on the new peer input data. An important point is that this model cann't be used directly as it supposes the availability of a global view of the DHT status, so that we can calculate all possible [solutions to the ILP]/[values of all criteria]. But it serves as a reference to the optinal solution we can get. All other contributions are based on that model. We will see after for the two next contributions how can we overtake this obstacle by managing the DHT to consider intra-DHT criteria or by building a partial view sufficient enough to get the optimal solution in short time or at least a good solution for a specific extra-DHT criteria.

Related Work

We are not aware of any related work at the general level, or any work that address new peers insertion while ensuring multiple properties. For fast building DHTs, many works have been done on the parallelization of new peers join to do parallel joins starting from a specific topology that connects the peers before deciding to bootstrap the DHT. Clearly those algorithms don't address efficiency of the build DHT and also don't lead to operational DHT in the case where peers have their input data. Also those works can be used only in the bootstrap phase and don't consider future new peers joins.

1.3.2 Load- and traffic-balancing

We propose the use of a distributed structure based on a *skip list* so that it will manage the DHT to facilitate access to the join peer [that satisfies the most of efficiency properties]/[to have efficiency]. The Skip List can be seen as practical solution to the formal model to consider criteria that depends only on the DHT status (intra-DHT criteria).
As an demo the skip list has been used to ensure both load- and traffic-balancing while building the DHT. Load-balancing ensure] that the quantity of data each peer have to manage is approximatively the same for all peers. Traffic-Balancing consider the number of messages each peer receive or forward, while new peers are joining the DHT or users are querying for data items. Those two criteria have an important impact on the DHT efficiency.

Related Work

Lots of work have been done to ensure load-balancing[**?**] and traffic-balancing[**?**] that we don't want to detail here. Very briefly, to ensure load-balancing, almost all solutions fold in two categories. In the first category over-loaded peers have to find underloaded ones to send to them some of their load. The main difference with our solution is that we address the problem in the building phase while those solutions address it after when using the DHT. So we can consider that our solution and others are complementary. Also we don't transfer data to ensure load-balancing we just take profit from the mandatory transfers when the status of the DHT changes. The other category introduce the notion of *virtual peers*. The idea behind is instead of moving data directly between peers, if a peer detect that it is overloaded it creates a new peer; the virtual peer, that will be responsible for a part of its data and will be hosted on the machine where an underloaded peer is running. for those solutions our

work can be well used even when the building is terminated as the creation of virtual peer is considered as a regular peer join operation.
A closest work to our was proposed by by XXXX in [**?**], where they use a tree-based overlay augmented by a skip list to redirect the new peers to the overloaded peers. This work was applied only on specific overlay/DHT (BATON) and consider only one property (load-balancing) where our solution is general can consider several properties based on the formal model we first define.

1.3.3 Data Transfers Minimization

When building DHTs, each new peer insertion induce uncontrollable data transfers due to the changing of the configuration of the DHT and inout data of new peer that it want to store in the DHT. Minimizing those transfers is important to fast build operational DHTs: DHTs that are ready to use and all data stored in is accessible for users.
Following the model we introduced in contribution 1.3.1, data transfer minimization is an extra-DHT criterion. In fact, it depends on inout data of each peer. –**this parag**–] In our best knowledge no work has been done to consider this criteria. Even more only very few works have been done on extra-DHT criteria, we can cite the criteria of *network-locality* that we will discuss in related work.
So following the built procedure, a new peer will look for the join peer from the DHT that will induce less transfers considering its input data. This will need a global knowledge that doesn't exist, unless it contacts all DHT peers which is too costly. Considereing that we introduce an algorithm that build a partial view that is more intersting than the local one. After that we exploit this partial view to develop a heuristic that minimize new peers data transfers after joining the DHT. The heuristic in the best case find the the optimal join peer to insert the new one, and in the worst case have have a very low overhead. We also provide a complexity study of the heuristic execution time and our experimental results.

Related Work

In our best knowledge no work has been done on data transfer minimization when building DHTs. But many related work have been done to consider some criteria that are close to our as *content-Locality* and *network-locality*. Content-locality [**?**] ensure that each peer will store locally its input data. This was proposed mainly for data confidentiality issues as well as path-locality[**?**]. Content-locality is a criteria that is stronger than data transfer minimization in case where if each peer keeps its input data there will be no transfers. Unfortunately the resulted overlay is not a DHT but just an indexing overlays. So those works can't be adapted to our criteria in the context of Building DHTs. Also those solutions don't deal with efficiency issues like load-balancing.
An other criteria that is similar to our, in the way that is also an extra-DHT criterion (i.e it depends on both DHT status and new peer characteristics) is Network-locality. Ensuring network-locality requires that the build DHT neighborhoods will approximatively reflect the physical network links, in the goal to reduce latencies of messages exchanged between neighbor's peers. This imply for each new peer, at the building phase, to find the join peer that is physically close to it among all DHT peers. **apparently,**] the criteria of network-locality

is well studied. Some proposed solution based on groups are mainly centralized[**?**]. Other need a heavy maintenances of the groups[**?**]. There is also very efficient and distributed solutions[**?**], but unfortunately are problem-specific, and relay on the hiearchy pf IP addresses to define proximity classes. An other point that make data transfer minimization more challenging is that data changes over time which is not the case for IP addresses that are fixed for each peer. Other differences that make hard the adaptation of solution proposed for network-locality are discussed in details in [**?**].

1.4 Project Management

1.5 Organization

The chapters of this report are organized as follows :

- Chapter 2 provide a general formal definition of DHTs that we will base on to develop our algorithms. It also presents Content-Addressable-Networks (CAN) as an example of DHTs that we will refer to all over this report. Finally the chapter presents the canonical algorithm used to build DHTs.
- Chapter 3 Describe the proposed formal model to build efficient DHTs in the general case. This is done by considering several properties to ensure to have performances.
- Chapter 4 Based on the model described in the previous chapter, this chapter presents a practical solution that uses a distributed structure (a Skip List) to manage the DHT in a way to facilitate join of new peers while ensuring properties related to the DHT. This structure was used, as a demonstration, to ensure Load- and traffic-balancing together.
- Chapter 5 shows how can we build a partial view of the DHT, that is more interesting than the local one, knowing that the DHT is a distributed structure over all peers. the chapter also present how can we exploit the partial view to minimize data transfers when building the DHT.
- Chapter 6 presents briefly the implementation of the algorithms presented in this report and the simulation tools used to validate our results.
- Chapter 7 provides a conclusion and points to future work on the topic.

2

Preliminaries

This chapter present a reference definition of structured overlays (and thus DHTs included) that we will base on to develop our algorithms. It also describes briefly *Content-Addressable-Networks* or CAN as the example of DHTs that we will refer allover this report for giving examples or for practical evaluations. Finally the chapter reintroduces] how to build DHTs referring to the presented definition of DHTs.

2.1 Reference Definition of Structured Overlay Networks/DHTs[?]

In any overlay a group of peer $\mathscr{P}$ provides access to a set of resources $\mathscr{R}$ (data items in our case) to an (application-specific) virtual identifier space $\mathscr{I}$ using two functions : $\mathscr{F}_P : \mathscr{P} \longrightarrow \mathscr{I}$ and $\mathscr{F}_R : \mathscr{R} \longrightarrow \mathscr{I}$. These mapping establish an association of resources to peers using a closeness metric on the identifier space. To enable access from any peer to any resource a logical network is built, i.e., a graph is embedded into the identifier space. These basic concepts of overlay networks are depicted in Figure **??**.
Par Abus de langage], virtual identifier space <=> virtual space <=> identifier space.

2.1.1 The virtual identifier space

A central decision in designing an overlay network is the selection of the virtual identifier space $\mathscr{I}$ which has to possess some *closeness metric* $d : \mathscr{I} \times \mathscr{I} \longrightarrow \mathbb{R}$, where $\mathbb{R}$ denotes the set of real numbers. d must satisfy properties 1-3 bellow and if possible should satisfy properties 4-5.

$$\begin{array}{l} \forall x, y \in \mathscr{I} : d(x, y) \geq 0 \\ \forall x \in \mathscr{I} : d(x, x) = 0 \\ \forall x, y \in \mathscr{I} : d(x, y) = 0 \Rightarrow x = y \\ \forall x, y \in \mathscr{I} : d(x, y) = d(y, x) \\ \forall x, y, z \in \mathscr{I} : d(x, z) \leq d(x, y) + d(y.z) \end{array} \tag{2.1}$$

If d satisfies all the five properties then $(\mathscr{I}, d)$ is a metric space. However, in many cases only the first three properties will be satisfied. In this case we will call $(\mathscr{I}, d)$ a *pseudo-metric space*.

2.1.2 Mapping peers to the identifier space

The mapping $\mathscr{F}_P : \mathscr{P} \longrightarrow \mathscr{I}$ associates peers with a unique virtual identifier from $\mathscr{I}$. Different approaches can be distinguished by the properties of the chosen functions $\mathscr{F}_P$:

- *Completeness:* $\mathscr{F}_P$ may be complete or partial. When Fp is partial, peers might (temporarily) not be associated with an identifier.
- *Morphism:* If no replication (for fault-tolerance) is required, $\mathscr{F}_P$ will be one-to-one (injective), i.e., $\forall p, q \in \mathscr{P} : p \neq q \Rightarrow \mathscr{F}_P(p) \neq \mathscr{F}_P(q)$.
- *Dynamicity:* $\mathscr{F}_P$ can be either statically defined , e.g., by its physical address or other unique attributes, or dynamically change over time. **In order to simplify our notations, in the following we will focus on the structural aspects and will not explicitly represent time-dependency in our notations.]**

Additionally, $\mathscr{F}_P$ may satisfy certain distributional properties, for example, that the range of values of $\mathscr{F}_P$ follows a certain distribution in space I, e.g., uniform. Such properties may then be exploited, for example, for load balancing. The properties $\mathscr{F}_P$ satisfies will be denoted as $\mathscr{C}_{FP}$ in the following. **In our work]**, we consider $\mathscr{C}_{FP}$ as $\mathscr{C}_{FP} = \{$ complete, injective, -dynamic- $\}$.

2.1.3 Mapping resources to the identifier space

The mapping $\mathscr{F}_R : \mathscr{R} \longrightarrow \mathscr{I}$ associates resources with identifiers from $\mathscr{I}$. The choice of this mapping can be critical for the application using the resources. If the resources should be identified uniquely, $\mathscr{F}_R$ has to be injective. The distribution of identifiers generated by $\mathscr{F}_R$ has an important impact on the load-balancing properties of the overlay network embedded into the space $\mathscr{I}$.
A standard example for $\mathscr{F}_R$ and $\mathscr{F}_P$ is a uniform hashing function. This will generate a uniform distribution of peers on the identifier space and implicitly provides load-balancing as also the resource identifiers are uniformly distributed. However, clustering of information will not be possible and thus higher-level search predicates such as range queries will be expensive to process.
Our work], is especially for DHTs that use non-uniform hash functions.

2.1.4 Management of the identifier space

At any point in time, $\mathscr{I}$ is managed by the set of current peers $\mathscr{P}$. The responsibility for peers for specific identifiers is captured by a function $\mathscr{M} : \mathscr{I} \longrightarrow 2^{\mathscr{P}}$, which associates

with each identifier of a resource r, $i = \mathscr{F}_R(r) \in \mathscr{I}$, the set of peers that are managing r. Through $\mathscr{I}$, each peer p is assigned responsibility for the set $\mathscr{M}^{-1}(p)$ of identifiers. Locating a resource r corresponds to finding a peer in $\mathscr{M}(\mathscr{F}_R(r))$. The lookup operation of overlay networks typically provides an implementation of $\mathscr{M}$ through routing. We may identify various basic properties for $\mathscr{M}$:

- *Completeness:* $\mathscr{M}$ may be complete or partial. When $\mathscr{M}$ is incomplete, identifiers might (temporarily) not be associated with a peer. Typically the mapping will be complete, such that each point of the identifier space is under the responsibility of some peers, i.e., $\forall i \in \mathscr{I} : \exists p \in \mathscr{P} : p \in \mathscr{M}(i)$.
- *Cardinality:* To provide fault-tolerance, $\mathscr{M}$ typically contains more than one element, i.e., a set of peers is responsible for managing each identifier.
- $\mathscr{M}$ *induced by proximity:* A standard way to specify $\mathscr{M}$ is that identifiers are associated with their closest peers, i.e., $p \in \mathscr{M}(i) \Rightarrow d(\mathscr{F}_P(p), i) = min_{q \in \mathscr{P}} d(\mathscr{F}_P(q), i)$.
- *Dynamicity:* $\mathscr{M}$ typically changes dynamically as the set of peers and their mapping to the identifier space changes.

In the following $\mathscr{C}_M$ denotes the properties $\mathscr{M}$ satisfies.
When $\mathscr{M}$ is **injective]** and complete, we can see it as a *partition* of the identifier space $\mathscr{I}$, where each peer p is responsible on a part of $\mathscr{I}$ that we call its *responsibility zone* $Z_p = \mathscr{M}^{-1}(p)$. As $\mathscr{M}$ is dynamic, We will have for any time :

- $\bigcup Z_p = \mathscr{I},\ p \in \mathscr{P}$, because $\mathscr{M}$ is complete,
- $\bigcap Z_p = \varnothing,\ p \in \mathscr{P}$, because $\mathscr{M}$ is injective.

2.1.5 Graph embedding

An overlay network can be modeled as a *directed graph*, $G = (\mathscr{P}, \mathscr{E})$, where $\mathscr{P}$ denotes the set of vertices (i.e., peers) and $\mathscr{E}$ denotes the set of edges. Due to the dynamics in overlay networks, G is time-dependent, but we will not explicitly denote this. By virtue of this graph we define a neighborhood relationship $\mathscr{N} : \mathscr{P} \longrightarrow 2^{\mathscr{P}}$, such that for a given peer p, $\mathscr{N}(p)$ is the set of peers with which peer p maintains a connection, i.e., there is a directed edge (p, q) in $\mathscr{E}$ for $q \in \mathscr{N}(p)$. The properties of the overlay network relate to properties of the directed graph generated by $\mathscr{N}$ and to the properties of the embedding of the graph into the (pseudo-) metric space $(\mathscr{I}, d)$. We can list :

- *Uniqueness:* For deterministic systems, e.g., Chord[**?**], DKS[**?**], for a given set $\mathscr{P}$ and mapping $\mathscr{F}_P$ only one valid network $\mathscr{N}$ exists. In randomized systems such as P-Grid[**?**] and randomized Chord[**?**], multiple valid $\mathscr{N}$ are possible.
- *Graph diameter:* A small diameter provides lower bounds on the latency of routing in the network.

- *Connectivity:* Some overlay network approaches may require that the overlay graph is connected at any time.

The properties $\mathscr{N}$ satisfies are denoted by $\mathscr{C}_N$ in the following. At this point we are able to completely characterize the structural aspects of overlay networks by the following definition:

Definition. The structure of an overlay network $O \in \mathscr{O}$ for a set of peers $\mathscr{P}$ is given by $O = (\mathscr{I}, d, \mathscr{F}_P, \mathscr{C}_{\mathscr{F}_P}, \mathscr{M}, \mathscr{C}_{\mathscr{M}}, \mathscr{N}, \mathscr{C}_{\mathscr{N}})$.

For the rest of the report we consider structured overlays (DHTs) that have the following characteristics :

- The mapping of peers to the identifier space $\mathscr{F}_P$ is complete, injective and dynamic.
- The mapping of resources to the identifier space $\mathscr{F}_R$ is non-uniform (exactly, order-preserving to have clustering)
- $\mathscr{M}$ is complete, injective and dynamic. Therefor we consider $\mathscr{M}$ as a partition of $\mathscr{I}$.
- The embedding graph $\mathscr{N}$ is unique and is connected at any time.

2.2 Building DHTs

So referring to the definition XX, we can say that building DHTs consists of building a partition $\mathscr{M}$ of the virtual space $\mathscr{I}$ considering current peers. As we have seen before, the building of DHT is incremental by insertion new peers. The first peer p_1 will manage all the virtual space, **i.e.,** $\mathscr{M} = \{(p_1, \mathscr{I})\}$. As well as new peers arrives they select a random identifier in the virtual space. This identifier will decide of the join peer; the peer who will insert the new one by ceding a part of its zone; and also the part of the join peer zone it will get. After having all peers inserted, $\mathscr{M}$ will capture all responsibility zones and the peers associated.

2.3 CAN : Content-Addressable Networks

Many DHTs have been proposed[**?**], where they differ in one or more aspect of our reference definition. In this report we will use the *Content-Addressable-Networks* (CAN) DHT as a reference example, and to implement our algorithms for validation.

Virtual identifier space $\mathscr{I}$. CAN uses a k-dimensional Euclidean space with virtual identifiers being coordinates in the space. The distance function d is the Euclidean distance.

Management of the identifier space $\mathscr{M}$. At any time the virtual space $\mathscr{I}$ is divided between current peers into responsibility zones, where all peers zones forms a partition of $\mathscr{I}$. Each peer p is responsible on data items having identifiers in its zone Z_p. This partitioning

is dynamic and depends on the number of peers participating in the DHT. When new peers join the DHT it became responsible of a zone that first was belonging to the DHT peer who did the insertion (join peer).
CAN uses a special algorithm to maintain $\mathscr{M}$, each time a new peer p_1 want to be inserted, the join peer p_j will cede the half of its responsibility zone by dividing it over one dimension. Next time a new peer p_2 want to join the DHT from any of peers p_1 or p_j, they have to divide their zones over another dimension different from last partitioning dimension. This makes the division of the virtual space alternate over the k dimensions. This is important to have low routing latencies as we will see just after. Figures XX and XXX show an example of partition of 2-Dimensional and 3-Dimensional CAN.

The connectivity Graph G. The CAN neighborhood relationship $\mathscr{N}$ forms a *k-torus* structure, where two peers are neighbors if their responsibility zones **spans overlap along k-1 dimensions and abut along one dimension]**. Figure XX illustrates this topology for a 2-dimensional CAN.

Routing algorithm. CAN uses a greedy algorithm for routing messages. Each message has an identifier, and is forwarded at each peer to the closest neighbor, until reaching the responsible peer. For a k dimensional space partitioned into n equal zones, the average routing path length is $(k/4)(n^{1/k})$ and individual peers maintain $2k$ neighbors.

3

Model for Building Efficient DHTs

In this chapter we will show that building efficient DHTs needs to consider multiple criteria. We will first start by considering a very known criterion : load-balancing, especially where the DHT data distribution is not uniform. By doing so, we will notice a load-imbalance, but this time, concerning the number of messages routed by peers. Dealing with that, is to consider *traffic-balancing*. For that we introduce a model that consider multiple criteria while building DHTs, based on the reference definition of a DHT presented in previous chapter.

3.1 Philosophy

As building DHTs is incremental by inserting each time new peers, optimize the building phase relay principally on optimizing the join operation. So we will work on how to insert new peers while considering some criteria.
Our philosophy on peers insertion is that, when new peer want to join the DHT, it's actually asking for/getting a responsibility zone from the virtual space. So each DHT peer will propose to the new one two choices; the two sub-zones of its zone; as, if it will be [selected as]the join peer it will divide its zone and the new peer can select one of the two resulting sub-zones.
Therefor we **resume]** peers insertion as, from one side, DHTs peers that propose possible responsibility zones for a joining peer, and for the other side, new peer that have to make one choice from all proposals. This choice of the responsibility zone will [be also, indirectly, a choice of the join peer]/[decide itself on the join peer]. In some cases where a criterion depends only on DHT peer characteristics, we may talk directly about choosing the join peer, as this criterion will be indifferent relative to its two proposed sub-zones.

3.2 Load-balancing

Most of existing] load-balancing algorithms imply moving data from one peer to another. **But as in DHTs, we don't have control on data and as each data item has a unique identifier, we use the notion of virtual peer]**. In fact, when a new peer gets its responsibility zone from the join peer, it will also receive data mapped to that zone from the join

peer as it is no more responsible on those data items. What we want to do is to take profit from data transfers induced by new peer insertion to ensure "some degree" of load-balancing.

Therefor, to be inserted, each] new peer have to select the join peer with the higher load to receive some of its data and thus reduce its load. Note that in this context, we select directly the join peer and not sub-zone because this criteria (load-balancing) concern the join peer.] This can be formulated as an ILP[1] problem, where the objective function to maximize is the load of a DHT peer:

$$\begin{cases} Min\ L = \sum_{p \in \mathscr{P}} (\gamma_p . L_p) \\ \gamma_p \in \{0,1\}\ and\ \sum_{p \in \mathscr{P}} \gamma_p = 1; \\ where\ L_p\ is\ the\ load\ of\ peer\ p \\ \mathscr{P}\ is\ the\ set\ of\ current\ DHT\ peers \end{cases} \tag{1.1}$$

We made use of boolean coefficients γ_p, to impose one choice of the join peer. ILPs are known as NP-hard problems, but as we have a limit number of solutions (number of DHT peers) we can test all of the possible solutions -je veux dire qu'on a pas besoin de resoudre le systeme lineaire-].
To get all possible solutions, we need a global view of the DHT status. As the DHT is a distributed structure, no one peer can provide this global view. For first time we will accept] to use a central DHT which will keep track of all DHT peers and their current load. After that, When a new peer want to join the DHT, it will contact this directory to get the most loaded peer. This is only a temporary solution as it is centralized: the directory will have a huge load to manage (information storage, peer load updates, new peer requests, ..), especially when the number of peers become considerable, so it is not a *scalable* solution.

By using this solution on small DHTs, we will notice that load-balancing is well ensured, and peers tend to share data load between them. Figure XX shows an example of DHT with data mapped in the virtual space and the responsibility zones of participating peers. We can see clearly that peers are concentrated in areas of virtual space where there is more data.

3.3 Traffic-balancing

When DHT data distribution is very skewed, considering load-balancing will lead to another problem that impact/concern also DHT performances. If we take -si on reprend-] the example shown in figure XXX, we will see that there is some peers with big responsibility zones that have a lots of neighbors. Figure XXXX depict those peers. In normal usage of the DHT, those peers will routes lots of messages as they form like "bridges" between areas

[1] Integer Linear Programming

of the virtual space with considerable amount of data. This will have a negative impact on DHT efficiency as those bridge peers will form bottlenecks, and will be overloaded by messages, will have high computation load to find routes to forward messages, and all this imply high bandwidth consumption.

It's important to notice that the problem of traffic-balancing is independent from load-balancing in the general case. As even if the virtual space is uniformly divided between peers and data distribution is uniform, some data items may be more popular than others and will imply high traffic load for some DHT peers.

To consider traffic-balancing we can well use a similar solution to load-balancing, by defining another objective function that ensure this criteria. For example find the peer with more neighbors or the peer with the big number of messages it received or forwarded, etc ... Nevertheless, we will not deal with load-balancing]. So we need to find a model that consider both load- and traffic-balancing and also other possibly needed criteria. This is what we will present next section.

3.4 General model [for optimal insertion]

We found that many problems can impact DHT performances, and considering them at the building phase will lead to more efficient ready-to-use DHTs. We have seen how to formulate the problem of new peers insertion while considering one criterion in the previous section. This can be generalized to consider multiple criteria as follows :

$$\begin{cases} Min\ \Delta = \sum_{i=1}^{k} (\rho_i . C_{i,z}) \\ \\ where\ k\ the\ number\ of\ criteria \\ \quad C_{i,z}\ criterion\ C_i\ evaluated\ for\ sub\ zone\ z \\ \quad \rho_i\ coefficient\ of\ criterion\ C_i \end{cases} \tag{1.2}$$

So when a new peer want to join the DHT it selects the responsibility zone (or the join peer that satisfies most of the specified criteria considering the priority specified by coefficients. This model, in addition to be simple and easy-to-use, has the following characteristics:

- *general:* consider any number of criteria, whether they are related to peers or zones. It is important that the model can consider criteria related to peer characteristics. Because in some situations, e.g., DHT data distribution is uniform but peers are heterogeneous (different computing power, different connexion bandwidth, ...) and we need to consider these differences,
- *parameterizable:* by associating coefficient to criteria. Even more the objective function can be generalized to non-linear function (as we don't aim] to resolve the system),

- *dynamic:* as it concerns each new peer insertion, the future choice of the join peer will depends on the current DHT status, and so it reacts to DHT changes.

On criteria types. In our model, we combine all criteria in the objective function. This was our key idea to consider multiple criteria. However, in practice some criteria are more challenging than others as their evaluation [requirements] differs [radically]. We distinguish two types of criteria based on their relation to the DHT and the new peer:

- *intra-DHT criteria:* their evaluation depends *only on the DHT status*, like load-balancing and traffic-balancing as they concern information related to DHT peers. Being independent from new peers, those criteria can be calculated even before new peers arrives,
- *extra-DHT criteria:* their evaluation depends on the DHT status, and *also on new peer characteristics*. Thus their evaluation change for each new peer even if the status of DHT doesn't change, like network-locality, or new peer data transfers minimization. This is problematic as those criteria can't be calculated only when the new peer arrives, eliminating then any possible pre-calculation, and so putting much **exigences]** on insertion time.

Next step : putting all in practice. Now having our model formulated, to use it in **practice]** for large DHTs, we need to **s'en passer]** of the directory, as it is not scalable, not fault-tolerant and will represent a bottleneck for the whole DHT. So in next chapters we will see principally how to get possible solutions in a distributed and scalable manner, i.e. without relaying on a central entity. This for intra-DHT criteria (in chapter 4) and some of extra-DHT criteria (in chapter 5).

4

Load- and Traffic-balancing

To consider criteria that depends on the DHT status (intra-DHT criteria) when inserting a new peer, we need to provide the informations about DHT status and facilitate the access of new peers to that informations. So in this chapter, we will first present almost all possible solutions to manage the DHT to provide the required informations. Second we will focus on the solution we adapted: a distributed data structure based on a skip list, and present its advantages and how to use it. Finally we provide our experimental result using the skip list for, as example, considering load- and traffic-balancing, we describe also experiments setup.

4.1 Managing the DHT

Based on the model defined, to insert a new peer considering intra-DHT criteria, each DHT peer will calculate the cost (value of the objective function) of its proposal. Then the new peer will choose the optimal (minimal or maximal) one. As no peer has a global view of the DHT, many distributed solution can be proposed to facilitate the access of the new peer to those information:

- Store the information *in the DHT* itself as *meta-data:* this is the first intuition as the information about the proposal of each peer is distributed, and we want to get it from any peer of the DHT (as the new peer will first get randomly a peer of the DHT for its insertion). So the DHT itself will be the best place where to store those information. Nevertheless, this solution will faces the key/value constrained] model used to manage data. In fact in a DHT, as we have seen before, each data value has its key that we use to store it and later to fetch it. And the challenge now, is how this information can be identified? which key will be used for each peer proposal? how the new peer will get those data? because if it asks for each proposal one by one the insertion will take lot of time and can't be acceptable. Else, is there a way for the new peer to ask just for the optimal proposal directly?
- Use *Gossip algorithm:* a very common solution to disseminate distributed information is to use *gossiping*. As its name suggests, a gossip algorithm will relay on the connectivity graph of the distributed system (the DHT in our case), that is supposed to be connected, to forward the information using neighborhood relationships. In our context, each peer

will first send its proposal to its neighbors (or a subset of its neighbors for bandwidth optimization). When a peer receives a proposal from a neighbor, it stores the proposal locally and forward it to the other neighbors, as it did for its own proposal. After a stabilization time, all proposals will be available locally at each DHT peer. This is also known as flooding algorithm. The problem with this kind of solutions is their high bandwidth consumption, as they generate a lot of messages, and the accuracy of the collected informations. Indeed when the system is very dynamic those solution are inefficient, because most of time the received information will be outdated very rapidly as the status change regularly. And if we want to get the new information quickly, we will have to accelerate the flooding by augmenting the retransmission rate pushing then a high traffic in the network. So considering Building DHTs that induce high dynamism as there may be a lot of parallel joins, use gossiping reveals inappropriate, as most of proposals each peer will store locally, will be outdated.

- Use a *distributed structure:* In a more organized manner, we may use a distributed structure. As a DHT is, a distributed structure controls the interactions between participants and all exchanged information. This is done by defining a set of algorithms and invariants that each participant must respect. However it has some additional costs than other solutions: starting from the conception phase to maintenance of the structure, but, if it's well designed considering the problem specificities, it will be more efficient and have more guaranties. So in our context, we need a structure that can regroup all peer's proposals, and the more important, facilitate for the new peer the access to the optimal proposal in a short time. This is the solution we adapted. the question now is what kind of structure is more appropriate to use? and which characteristics it must have? this what we will see after.

Distributed structure characteristics. In our context, not any distributed structure can be used, rather it must respect some characteristics:

- *Efficiency* : in time and space. The distributed structure must occupy only a small space on each peer. Typically constant or logarithmic relative to the number of peers, as the DHT may imply thousands of peers. Also for the same reason, the execution time of the operations of the structure have to be low. For distributed structure we measure the time complexity of an operation by the number of messages exchanged to achieve it. The typical complexity is $log(N)$ where N is the number of current peers. In our context, this will make the insertion fast and scalable **as many peers the DHT counts]**.

- *Self-stabilization* : a distributed structure is said to be self-stabilizing if after any execution of any number of its operations it will eventually converge to a legitime state. This is important for our structure, because there will be a lot of dynamism where peers can join an leave concurrently the DHT at their own peace. And the structure must have to adapt to those situation very quickly.

- *Fault-tolerance* : because faults in a distributed system are inevitable, we need to consider them **as part of the system]** when designing the distributed structure. In

DHTs, in addition to non predictable leaves and joins, peers may also fail, and we need that our structure still working even in the presence of failures.

4.2 Skip List as a Distributed Structure

4.3 Experimental Evaluation

5

Data Transfers Minimization

5.1 Data transfers types

5.2 Incoming Data Transfers Minimization

5.3 Outgoing Data Transfers Minimization

5.3.1 Building partial view

5.3.2 Take Decision based on partial information

5.3.3 Heuristic for outgoing data transfers minimization (ODMH)

5.4 Experimental Evaluation

6

Implementation

6.1 Tools & Environment

6.2 CANs

6.3 Skip List

6.4 ODMH

7

Conclusion

Bibliography