Fast Construction of Efficient Structured Peer-to-Peer Overlays

Lokman Rahmani

Fast Construction of Efficient Structured Peer-to-Peer Overlays

Student: Lokman RAHMANI
lokmanet@gmail.com - Master 2 IFI - CSSR
Supervisor : Fabrice HUET
fabrice.huet@inria.fr - INRIA - OASIS Team
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 handle distributed data in large networks. Our algorithms ca be used to build efficient DHTs with several properties or to recover fastly the last configuaration of a DHT after crash or exit.

In DHTs, the managed data distribution can be non uniform, and if we build the DHT using canonical algorithms it will create load imbalance between participating machines (or *peers*): it will take a lot of time and will lead to a DHT with poor performances. So to ensure efficiency in the general case, we first introduce a general easy-to-use model that can be used to consider several properties, that guarantee DHT performances. Based on that model, as a practical solution, we provide a distributed structure that permit to reorganize the DHT to facilitate the join of new machines, while considering the selected properties. As an example, this structure was used to ensure both load- and traffic-balancing over DHT machines.

To fast build DHTs, we need to have a global knowledge about DHT status, as we need to consider some properties related to the new peers characterstics. Nevertheless DHTs are distributed structures, i.e. each participating peer maintains a local state of the DHT which includes itself and a small number of its neighbors. No one peer have a global view of the whole DHT. To overcome this obstacle, we provide an algorithm that build a partial view of the DHT more interesting than the local one. 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

I truly feel privileged to have worked under the supervision of Dr. Fabrice Huet. He introduced me in the research area of distributed systems. He was available all the time and provide me advises and knowledge on both the theoretical and practical work. I would like also to thank all OASIS team members for their hospitality and rich discussions during my internship. In particular, I thank Amjad Alshabani, Iyad Alshabani, Laurent Pellegrino and Maeva Antoine. I would like to show my gratitude to Dr. Arnaud Legout who accepts to read and comment my reports. Finally, I would like to acknowledge all the academic team at Polytech'Nice Sophia and University of Nice Sophia-Antipolis that manages the IFI master and make it richer and more interesting.

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?[23]

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 also peers will have identifiers from the same virtual space. Then, each peer will be responsible on keys/values items that are close to its identifier 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[29, 19]. 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 a grid computing environment, the keys in a DHT can represent CPU power. Hence an application might query the DHT to search for all key in the interval 2000-4000 MHz. DHTs does not support naturally ranges queries. Over the last years, many works have been done to consider range queries [11, 15, 33, 25]. The Play project used the solution proposed by A. Datta and al. in [21]: 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 ensures 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. 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 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 identifier that is closest to its identifier. Once found, the insertion is done with the participation of 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 hosting peers. Figure 1.1 ilustrate an overlay network and the physical network 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, ...).

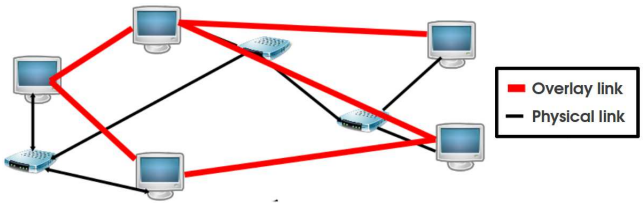

Figure 1.1: An overlay network built at the application level and the physical network it relays on.

As we will talk only about structured overlays, for simplicity we will omit the word 'structured' in structured overlays.

1.2 Applications on top of DHTs

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

Publish/Subscribe systems - In a Publish/Subscribe system there are two types of roles, as its name suggests : publisher and subscriber. Publishers are those who generate and provide contents. Subscribers are content consumers. A subscriber 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 subscribers for that topic.
It's clear that with the presence of many publishers and many subscribers that the system will have a huge load of content to receive, store and deliver at real-time. This becomes more difficult when publishers and subscribers are geographically distributed. This is the aim of the Play[5] European 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 [34]. Many other works have been done on using DHTs in Publish/Subscribe systems like [37]. Our 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 anywhere 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 (like consistency, availability, partition tolerance) and are known as key/value storage [22, 13, 20]. 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) [38], Host Identity Payload (HIP)[4], ...
- Web caching and web servers: Squirrel[26], DKS Organized Hosting (DOH)[28]
- In peer-to-peer file sharing applications : BitTorrent[16], Azureus, eMule, and eDonkey use the Kademlia DHT[31].

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 reference definition of DHTs[6], 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 has to find the best peer to be inserted 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 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 DHT. In the other side an extra-DHT criterion depends also on the new peer and its evaluation changes for each new peer. An example of an extra-DHT criterion is data transfer minimization where the quantity of data transfers will depends on the new peer input data.
An important point to note, is that this model can'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. But it serves as a reference to the optional 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[12, 17]. Clearly those algorithms don't address the efficiency of the built 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*[35] 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 model to consider criteria that depends only on the DHT status (intra-DHT criteria).
As a 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

[1] Integer Linear Programming

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 H. V. Jagadish and Al. in [27], 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 DHT (BATON) and consider only one property (load-balancing) where our solution is general can consider several properties based on the 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 the input 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 the input data of each peer.
So following the construction 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 a very low overhead. We also provide and discuss 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 [1, 40, 24] ensure that each peer will store locally its input data. This was proposed mainly for data confidentiality issues as well as path-locality[24]. 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[8, 41, 18, 30]. 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. The criterion of network-locality is well studied. Some proposed solutions based on groups are mainly centralized[41]. Other need a heavy maintenances of the groups[30]. There is also very efficient and distributed solutions[18], but unfortunately are problem-specific, and relay on the hierarchy 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 [3].

1.4 Organization

The chapters of this report are organized as follows :

- Chapter 2 provide a general reference 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 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 the 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 participating peers. the chapter also presents how can we exploit the partial view to minimize data transfers when building the DHT.
- Chapter 6 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 explain again how to build DHTs referring to the presented definition of overlays.

2.1 Reference Definition of Structured Overlay Networks[6]

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 2.1. In the rest of this report, we will use indifferently virtual identifier space, virtual space and identifier space.

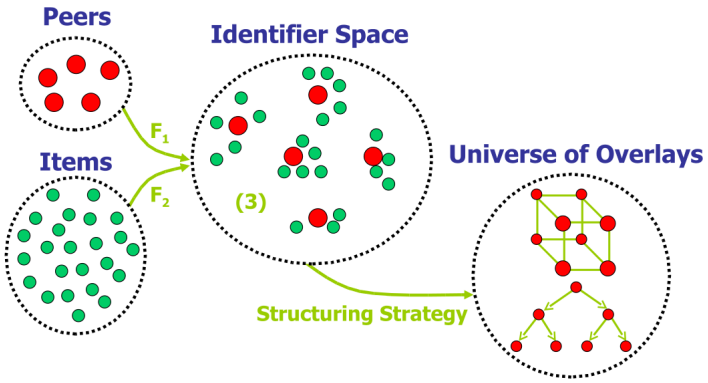

Figure 2.1: Reference definition of Structured Overlay network[10].

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.

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

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 concerns especially 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[39], DKS[9], 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[7] 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.

Building DHTs - So referring to the definition 2.1, 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.

2.2 CAN : Content-Addressable Networks

Many DHTs have been proposed (e.g. [39, 36, 9, 7, 31]), where they differ in one or more aspect of our reference definition. In this report we will use the *Content-Addressable-Networks*[36] (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. Figure 2.2 shows to examples of a 2- and 3-Dimensional CAN.

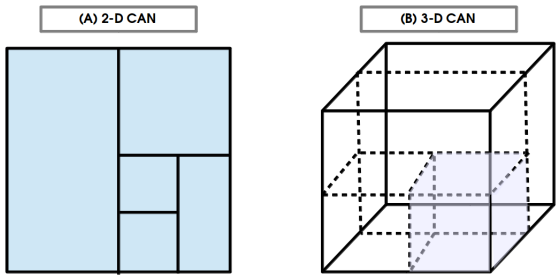

Figure 2.2: CAN virtual space example configuration.

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 2.3

how an example of partition of 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.

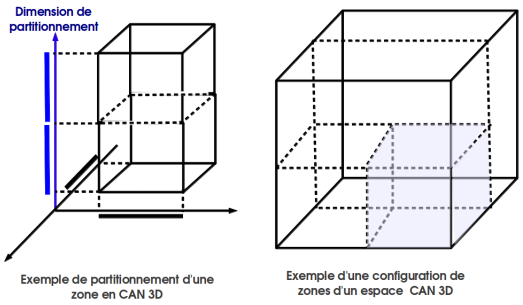

Figure 2.3: 3-D CAN virtual space partitioning.

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 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 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, DHT 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. 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

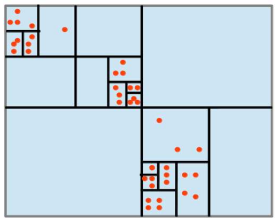

Figure 3.1: Load-balancing ensured. Peers are concentrated in the areas of space where there is more data.

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.
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 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 3.1 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 impacts also DHT performances. If we take the example shown in figure 3.1, we will see that there is some peers with big responsibility zones that have a lots of neighbors. Figure 3.2 depict those peers. In normal usage of the DHT, those peers will route lots of messages as they form like "bridges" between areas of the virtual space with considerable amount of data. This will have a negative impact on DHT efficiency as those bridge (or routing) 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.

[1] Integer Linear Programming

Figure 3.2: Traffic-balancing problems. Bridge peers have alot of neighbors and forward lots of messages.

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 no more consider 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 join

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, ...) 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 formally),

- *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 completely differ. 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 early computation, and so putting much constraints on insertion time.

Next step : putting all in practice - Now having our model formulated, to use it for large DHTs, we need to dispense 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 we 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 as an entry peer). 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 space 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.
- *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 legitimate 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 have to adapt to those situations very quickly.
- *Fault-tolerance* : because faults in a distributed system are inevitable, we need to consider them as a 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

So we will make use of a distributed structure to allow optimal insertion of new peers based on multiple (intra-DHT) criteria. The role of this structure is to let each DHT peer participate with its own proposal. And when a new peer arrives, it will use the structure to get the current optimal proposal and then, will proceed to its insertion. Typically the structure has to provide the following functions :

- $join()$: permit to a DHT peer to participate with its proposal,
- $leave()$: permit to a participant to withdraw its proposal,
- $update()$: permit to a participant to update its proposal,
- $getOptimal()$: permit to a new peer to get the peer who proposes the optimal value.

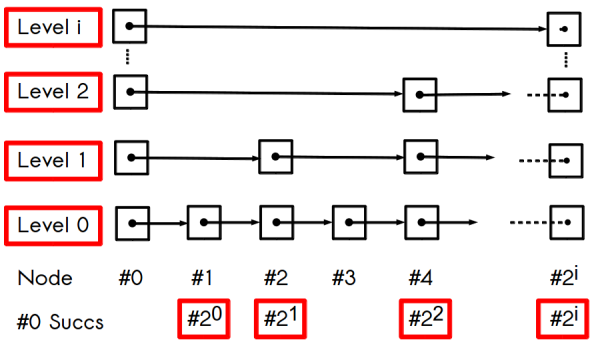

Figure 4.1: Skip links for element #0 within a perfect skip list

Skip list. A skip list is basically an *ordered doubly* linked list, where in addition to links to direct successor and predecessor, each skip list element (called node) have also other links to distant successors and predecessors, called *distant* links or *skip* links.
In the original version[35]; supposing that the number of nodes is known; skip links are calculated in such a way to ensure that searching any value in the skip list will take a logarithmic time relative to the number of skip list nodes. Indeed, each node will calculate a link to its 2^{ith} successor until reaching the last element. In the general case where the number of nodes is not known or changes over time, the author, in the same paper proposes a probabilistic version to ensure the same complexity with a high probability.

To implement the skip list, we used a deterministic algorithm, inspired from [32], to calculate skip links to have a *perfect* skip list. Figure 4.1 shows links that a node (node ♯0) must calculate to have a perfect skip list. Note that in the figure only successors links are shown for reason of clarity. Also we will use the term *neighbor* to designate at the same time a successor and a predecessor. So in the deterministic version, skip links are organized in levels. The first level contains link to to the first (or direct) neighbor. At each level i the skip links are pointing to a neighbor that is two times distant than the neighbor at level $i-1$. Therefor each level i calculates a link to the 2^{ith} neighbor until reaching the last node.

The skip list is well appropriate to implement our distributed structure as it have the required characteristics:

- skip list performances are mainly related to the establishment of the skip links. Rather than the probabilistic version, we use a deterministic version that converges always to the optimal configuration (a prefect skip list),
- as the skip list is a linear structure, its stabilization is more efficient. To ensure that we will use a set of algorithms similar to those used in Chord[39],
- an other interesting point about skip lists, is that the distant links can be also used for fault-tolerance, to recover from faults inducing many successive neighbors.

All details about algorithms used to exploit and maintain the skip list can be found at [2].

4.3 Experimental Evaluation

4.3.1 Experiments setup

To join the DHT each new peer first gets an entry peer from the directory that maintains a list of peers participating in the DHT. The DHT data counts more than 12 000 elements and the size of each one is between 5 and 50 Mega bytes. All tests were done using the SimGrid[14] simulator. We first implement a complete CAN DHT , and we implement on top of it the skip list algorithms. SimGrid have many API, we used the GRAS (for Grid Reality and Simulation) API, where the implemented application can be executed in simulation or on real machines.

4.3.2 Results & Discussion

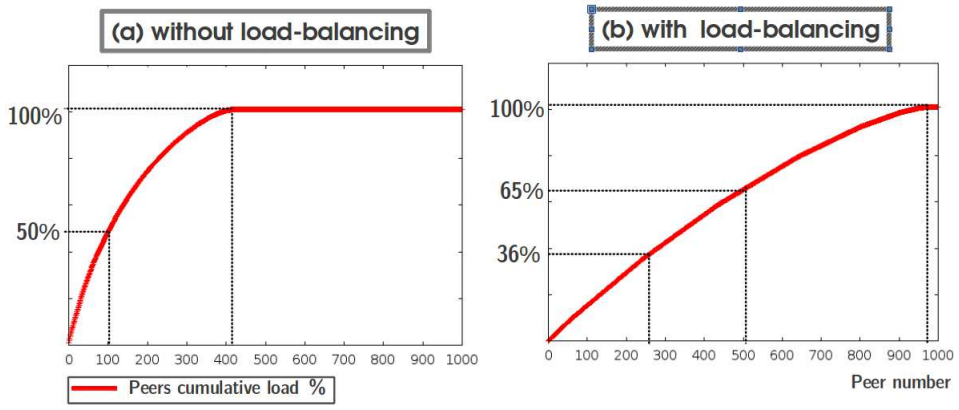

Figure 4.2: Skip list experiments for load-balancing

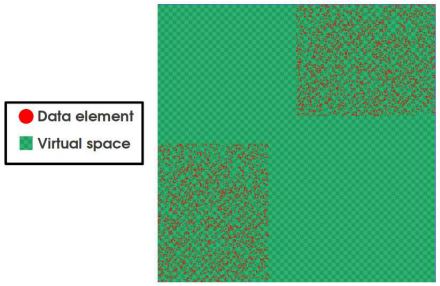

Figure 4.3: DHT data distribution

Figure 4.2 shows the data load of peers just after building the DHT while using the skip list to insert new peers considering load-balancing criterion comparing to the canonical join algorithm. The data distribution of the DHT is not uniform and is shown in figure 4.3. Each graph traces the cumulative data load of DHT peers, where peers are ordered in the X axis following the decreasing order of their load.
When using the canonical algorithm, graph 4.2.a shows that the total DHT data load is managed only by less than the half of peers, where the rest of peers have not practically any data load. However when using the skip list (to consider load-balancing criterion) graph 4.2.b shows that all peers participate and have approximatively the same load of data to manage.

5

Data Transfers Minimization

In this chapter we will see how to fast build DHTs by minimizing data transfers while inserting new peers. In the previous chapter we have seen how to insert a new peer considering multiple intra-DHT criteria in the general case. Nevertheless considering extra-DHT criteria, as new peer data transfers minimization, is more difficult. First due to the lack of global view of the DHT, and second the more challenging, that extra-DHT criteria depends on each new peer and can't calculated until the new peer arrives. So we will see how a new peer can build a partial view more interesting than the local available one. We then use it to develop a heuristic that lets the new peer minimize its input data transfers after being inserted. Yet the heuristic can't be used for general case. Rather it can be adapted to any extra-DHT criterion that its evaluation depends on available zones and has a bounded objective function. Finally, we present our experimental evaluation and we discuss the results.

5.1 Understanding Data transfers

After a new peer is inserted, there will be some uncontrollable data transfers as the configuration of the DHT has been changed: the responsibility zone of the join peer is narrowed, and the new peer became responsible of a zone of the virtual space. Therefor, first the new peer will receive data form the join peer that it's no more responsible on it. Second when new peer come with its data, it will send it to the responsible peers for storage.
Those transfers are qualified as *uncontrollable* as neither the new peer decides where it will store its input data, nor the join peer decides on which data it will pass to the new peer and which ones it keeps. Rather all depends on which sub-zone the join peer will keep and which one it will cede to the new peer, and on where each data element is mapped in the virtual space. Figure 5.1 shows this two types of transfers where we call, from the point of view of the new peer:

- *incoming data transfers*: transfers from the join peer to the new peer, i.e. data that the join peer is no more responsible on,
- *outgoing data transfers*: transfers from the new peer to the join peer and to other peers, i.e. new peer input data that doesn't map into its new acquired responsibility zone.

If we define the problematic of data transfers minimization when inserting new peer following our model (see section 3.4), we will have to resolve the following ILP problem:

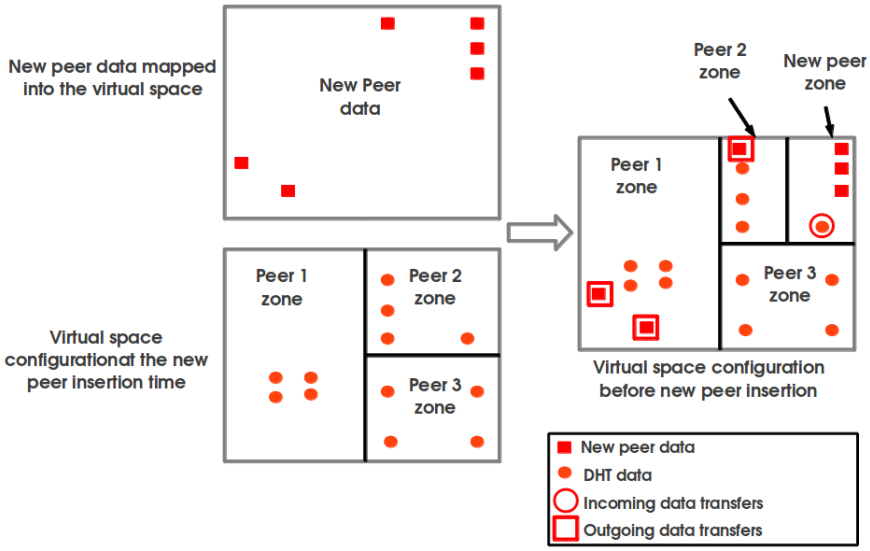

Figure 5.1: Data transfers when inserting a new peer

$$\begin{cases} Min\ T = \sum_{p \in \mathscr{P}} \gamma_p.(\alpha.T_p + \beta.T'_p) \\ T_p = Ti_p + To_p \\ T'_p = T'i_p + T'o_p \\ \gamma_p \in \{0,1\}\ and\ \sum_{p \in \mathscr{P}} \gamma_p = 1; \quad \alpha, \beta \in \{0,1\}\ and\ \alpha + \beta = 1; \\ where : T_p(resp.\ T'_p)\ are\ data\ transfers\ if\ new\ peer\ choose\ the\ first\ (rep.\ second)\ sub\ zone\ of\ peer\ p \\ Ti_p\ (To_p) :\ incomming\ (outgoing)\ data\ transfers \\ \mathscr{P}\ is\ the\ set\ of\ current\ DHT\ peers \end{cases} \tag{1.3}$$

So the new peer have to choose the sub-zone that will lead to less data transfers after its insertion. Note that for this criterion; differently than the criterion of load-balancing; we are talking about choosing the sub-zone not the join peer, as the criterion of data transfers minimization depends on the sub-zone that the new peer will get not the peer itself. Indeed, the sub-zone will automatically decide on the join peer.

In practice. It's clear that the criterion of data transfers minimization is an extra-DHT criterion as it depends also on the new peer (data). However, to propose a practical solution, we need to divide it in two sub-criterion:

- Outgoing data transfers minimization: concerns the new peer data t hat it will send to the join peer and other DHT peers. Or inversely, the new peer data that it will not keep after its insertion.

- Incomming data transfers minimization: concerns the data that the new peer will receive from the join peer, as it become under its responsibility.

One of the benefit of this separation is that the sub-criterion of incoming data transfers is intra-DHT as it doesn't depend on the new peer, but only on the data quantity each proposed sub-zone contains. This is interesting because we know how to deal with such criteria. In the next sections we will see how to consider practically each sub-criterion (or simply criterion).

5.2 Incoming Data Transfers Minimization

To consider incoming data transfers minimization criterion; being intra-DHT; we can well use our distributed structure to get the optimal solution to insert new peers. Even we can consider it with other criteria at the same time. For this case, each peer will have two proposals, where each sub-zone will be associated to the quantity of data it contains. The new peer have then to choose the proposal with minimum value. However, this criterion isn't very interesting alone, because if we consider it alone, it will minimize data transfers but leads to inefficient DHT. Indeed, new peers will choose sub-zones with empty data and there will be no share of load between peers.

We can think about considering this criterion with the criterion of load-balancing to ensure efficiency, but it still not interesting and non significant as the two criteria are completely contradictory. And considering load-balancing alone will be sufficient enough. In next section we will see how to consider outgoing data transfers minimization which is more interesting and more challenging also.

5.3 Outgoing Data Transfers Minimization

Minimizing new peer data transfers will lead to build *operational* DHTs very rapidly. We mean by operational DHT, a DHT where all data elements are accessible. Indeed when new peers join the DHT with considerable input data it will take a lot of time to store all the data as each data element need to be forwarded to the responsible peer. So as much the new peer can keep of its input data as the construction of the DHT will be faster.
To do so, the new peer need to get information about available sub-zones to get the one where it will keep the most of its data after it is inserted. As this criterion is extra-DHT, we need to provide those information in short time, as they will be calculated for each new peer. In this section we will see how to build a partial view of the DHT based on zone identification algorithm that provide information about some available sub-zones. After we will see how can we decide on the best sub-zone without the need of the knowledge about all available sub-zones, in some special cases. Using those intermediate results, we will present the heuristic we proposed to minimize new peer outgoing data transfers.

5.3.1 Building partial view

To properly describe the information about existing zones when new peer want to join the DHT, we need first to identify each zone (or sub-zone) uniquely so that the new peer can correctly choose the optimal one.

Most of DHTs overlays use the same partitioning procedure: the first peer will get the whole space, and when a new peer joins the DHT, the join peer (the peer that will insert the new one) will split its zone into two parts, and one will be given to the new peer. In fact we can define a deterministic algorithm to identify uniquely any available sub-zone at any time.

Note : In the following, all discussions will be given on CAN DHT as an example, but the ideas and solutions concern DHTs in the general case.

Zones can be identified using binary strings. The whole space will be identified by `0` or `1` following some convention, for example `0` if we start the partitioning over the first dimension (where there is multiple dimension like in CAN). After that we can identify all possible sub-zones using an inductive algorithm : for a zone identified by the bits string $b_0b_1...b_n$, after it is partitioned, its two sub-zones are identified by the bits strings $b_0b_1...b_n0$ and $b_0b_1...b_n1$. We can adapt a convention so that the choice of the identifiers for the two sub-zones will be deterministic. For example the sub-zone with lower values (for the partitioning dimension) will have the suffix `0`. Figure 5.2 show an example of a 2D CAN DHT with 5 peers, and the identification of responsibility zones of each peer over time.

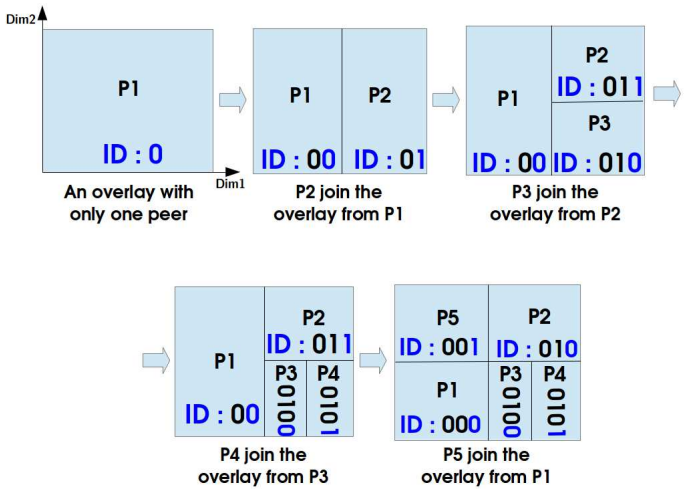

Figure 5.2: Zone identification algorithm. First we give an identifier for the whole space. After each time as a zone is divided new bits are added to create identifiers for the new sub-zones

This identification schema can be seen as a binary tree where each peer represents a zone. Each zone if partitioned will have its two sub-zones as children in the binary tree. Leafs are therefore the actual existing zones and other nodes are zones that no longer exist as they were partitioned before. Figure 5.3 shows an example of a 2D CAN overlay with 5 peers and the corresponding binary tree for zone identification.

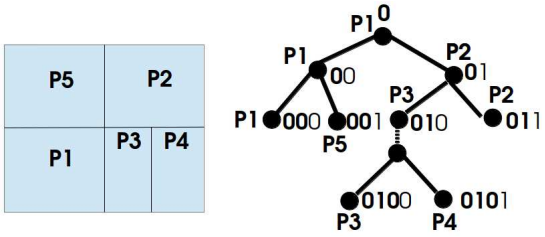

Figure 5.3: Zone identification schema tree. Leafs are actual zones managed by overlay peers, and intermediate nodes are old zones that were partitioned.

Using zones identifiers we can get some partial knowledge. Figure 5.4 gives an example on how to get a historical knowledge just by analyzing one zone identifier.

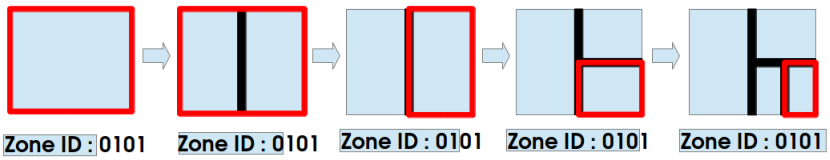

Figure 5.4: How to get a partial view from one zone identifier (0101). The first left bit (0) indicates the first dimension of partitioning, and by analyzing the identifier from left to right, each bit will indicate which sub-zone was partitioned until having the actual zone.

Those informations are historical but still important (especially where there is no leaves and failures) because they give an upper bound of data that the new peer can keep. The heuristic we propose is based on the use of this partial view to get information about the DHT status.

5.3.2 Take Decision based on partial information

We have seen that (in section 3.4), to insert a new peer in the optimal position considering an extra-DHT criterion, we need to calculate the objective function for each available sub-zone, and then select the one with the minimal value. This needs to have a global knowledge about DHT -to get all available sub-zones- that we haven't. However, using the zone identification algorithm we came to get a partial knowledge. So the question now is if we can made a decision on the optimal sub-zone based only on that partial knowledge: how can we take profit from it?
Noticing that, in our problem, that is considering new peer outgoing data transfers, the objective function has an upper bound (all new peer data will be transfered), and the new peer have to select exactly one sub-zone, we can decide that a sub-zone is optimal without calculating the objective function of any other sub-zone in one special case as the following lemma indicates :

Lemma: If for a given sub-zone the new peer will keep more than the half of its data, this zone is the optimal one considering the criterion of outgoing data transfers minimization.

Proof: lets recall how the virtual space $\mathscr{I}$ is managed by peers that we saw in the reference definition of DHTs (in section 2.1). 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. As we consider that $\mathscr{M}$ 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 $Z_p = \mathscr{M}^{-1}(p)$.
Lets consider SZ the set of all current available sub-zones. SZ forms also a partition of $\mathscr{I}$ as each two sub-zones proposed by the same peer are a partition of its responsibility zone. Lets define the function $T : SZ \longrightarrow \mathbb{R}$ that associates to each sub-zone z the quantity of data, $T(z)$, the new peer will keep if it selects z as its responsibility zone. Its clear that the function T is bounded by T_{MAX} the quantity of transfers where the new peer will loose all its data. So if the new peer will choose a sub-zone z' where it will keeps more than the half of its data, i.e. $T(z') \geq (T_{MAX}/2)$, z' is the optimal sub-zone it can get when looking to minimize its data transfers. Because, as the new peer can choose only one sub-zone from SZ, and SZ is a partition of the virtual space $\mathscr{I}$ any other sub-zone will keeps at most the half of its data.
Or more formally, we define the ILP problem that consider outgoing data transfers minimization as

follow:

$$\begin{cases} Max\ \mathbb{T} = \sum_{z \in SZ}(\gamma_z.T(z)) \\ \\ \gamma_z \in \{0,1\}\ and\ \sum_{z \in SZ} \gamma_z = 1 \end{cases} \tag{1.4}$$

If we have for a given sub-zone z', $T(z') > (T_{MAX}/2)$ and knowing that the objective function $\mathbb{T} \leqslant T_{MAX}$, we will have for the rest of sub-zones $\sum_{z \in SZ/\{z'\}}(\gamma_z.T(z)) \leqslant (T_{MAX}/2)$ as $\mathbb{T} = \sum_{z \in SZ/\{z'\}}(\gamma_z.T(z)) + \gamma_{z'}.T(z')$. So choosing z', i.e. $\gamma_{z'} = 1$ and for $z \in SZ/\{z'\}$, $\gamma_z = 0$, is the best solution to the ILP. Figure 5.5 illustrates that on an example.

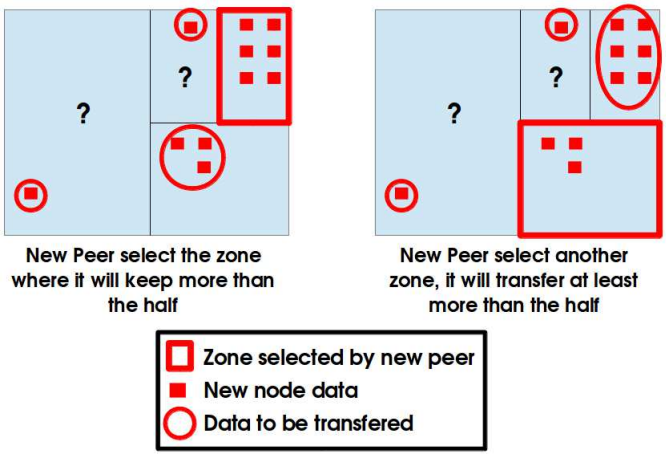

Figure 5.5: When minimizing new peer outgoing data transfers, the zone where the new peer will keep the half or more of its data, is the optimal one.

5.3.3 Heuristic for outgoing data transfers minimization (ODMH)

Now, having a partial view of the DHT built using zone identifiers, and the possibility to take a decision based on partial knowledge, we can propose a heuristic for the new peer so that it can choose the zone that minimize its outgoing data transfers.

In short, the idea of the heuristic is to get a partial view (about existing sub-zones) in the parts of the space where the new peer have most of its data. If the new peer data is very skewed; and then almost all data will be located in one place; the optimal solution will be very probably one of the sub-zones of the built partial view. Else if the new peer have its data located in different areas in the virtual space, the heuristic try to choose the best place to select and fetch for a good sub-zone. Finally if the new peer data is uniform over the virtual space, there will be no considerable gain between sub-zones, and we want to avoid useless calculation by choosing one sub-zone randomly to insert the new peer.

The heuristic proceed following next steps :

1. Be in the right position. Its clear that the partial partial knowledge we can get depends on the zones that we analyzed. So for the new peer, to keep most of its data it needs to get information about how the virtual space is organized in the area where most of its data is located.
To do so, the new peer will first do a lookup for the [point that represents the] centroid of its data and it will get as a response the zone identifiers of the responsible peer and of its neighbors. If we call new peer data set $Data_{NewPeer}$ where each *data* element is represented by the point where it is mapped

(x_{data}, y_{data}) and the size of the data (called the cost) $cost(x_{data}, y_{data})$, the centroid point $p_c = (x_c, y_c)$ will be calculated as follow:

$$\begin{cases} x_c = \sum_{(i,j) \in Data_{NewPeer}} \frac{i \times cost(i,j)}{|Data_{NewPeer}|} \\ \\ y_c = \sum_{(i,j) \in Data_{NewPeer}} \frac{y \times cost(i,j)}{|Data_{NewPeer}|} \end{cases} \tag{5.1}$$

Using the algorithm of zone identification the new peer can build a partial view more interesting than the one just containing just the entry entry peers and its neighbors responsibility zones. There will be two types of zones forming the partial view:

- Exact zone (EZ): the sub-zones of the entry peer and of its neighbors,
- Maximal zone (MZ) : the sub-zones that were calculated from zones identifiers,

The figure 5.6 shows an example of the partial view build from an entry peer and its neighbors zones.

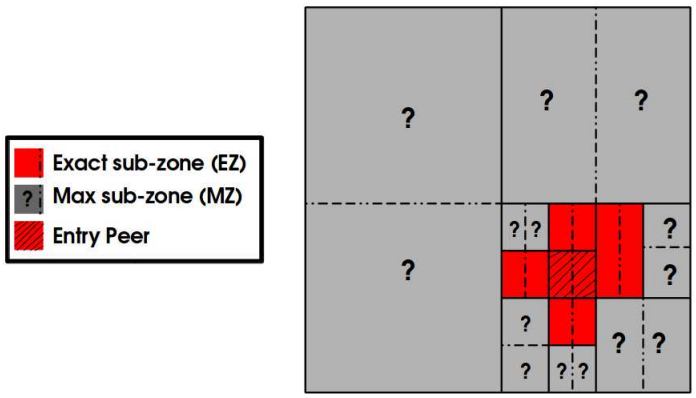

Figure 5.6: Partial view contains (a) exact sub-zones that new peer is sure that they exists (b) maximal sub-zones that the new peer didn't know how they are partitioned it have just the upper-bound of the maximal possible sub-zones that it can choose

2. Taking advantage from the partial view So the first step built a partial view of the available sub-zones in the area where the most of new peer data is supposed to be. Now the new peer will try to select the sub-zone where it will keep the most of its data, i.e. the sub-zone that contains more of its data. As we have seen, if an exact sub-zone contains more than the half of its data, its the optimal one, and the new peer will then send a join message to the peer that propose this sub-zone and the heuristic will exist. If no such sub-zone exists this can have one of two significations relative to new peer data:

- New peer data distribution is **uniform** (see figure 5.7.a): this imply that approximatively the same quantity of data will be mapped to each available sub-zone, and there will be no *noticeable* best sub-zone.
- New peer data is skewed but have **multiple clusters** (see figure 5.7.b) : this means that the new peer data is located in many parts of the space but not uniformly forming then many clusters of data, so the centroid falls between those parts (or clusters) where there is no or only few quantity of data.

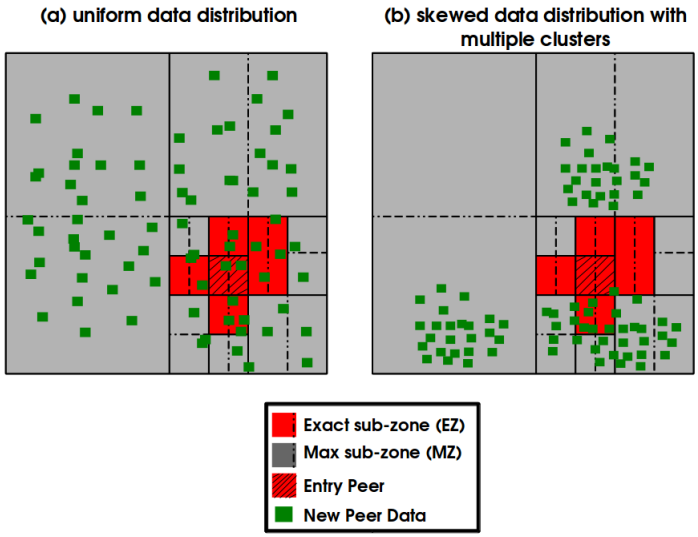

Figure 5.7: New peer is not in the right position : (a) New peer data is uniformly distributed, the expected gain will be very low (b) New peer data is skewed but contains multiple clusters, and it need to move to one cluster

So the challenge now is how to detect those cases, differentiate between them, and how the new peer will choose a sub-zone considering each case. This what we will see in next last two steps.

3. Detect uniform data distributions Its important to detect if the new peer data distribution is *considered* as uniform, because in that case all sub-zones will generate approximatively the same data transfers; or inversely keep approximatively the same quantity of data; and the possible gain between them will be negligible.

So the right question now: is when can we consider that the new peer data is uniform? Actually there is different ways to decide about, we will just present the one we used in our test. We tried to let the decision algorithm as parameterizable as possible. In fact, we first divide the virtual space in chunks with equivalent size, and we calculate how much data is mapped to each one. Then we calculate the *coefficient of variation* $C_v = \frac{\sigma}{\mu}$ which is a normalized measure of dispersion, where σ is the standard deviation, and μ is the mean. After we define a parameterizable threshold ϵ_{CV} that will determine if the new peer data distribution is considered uniform or not. If we have $C_v \leqslant \epsilon_{CV}$ this means that new peer data distribution is uniform and the new peer will then select randomly one sub-zone and the heuristic exits.

4. Dealing with skewed data with multiple clusters At this level it seems that new peer data is not well distributed over the virtual space and it contains two or multiple clusters (which is some how a distribution between the skewed and the uniform one). This means that new peer built a partial view not in the right place of the virtual space, and it needs very probably to move.
Moving is very costly (lookups and mapping) so the new peer needs to know first about the possible gain for moving and to decide if its really interesting to do so. Also an important point is that after moving the new peer can't go back to choose a zone from the last partial view: first because there will be no guaranty that the zone still exist, and second because the heuristic will have a high complexity.

To detect that case the new peer will test if the most of its data is located out of the exact zones. Also for this test we make use of ϵ a parameterizable threshold. Let MZ_{set} the set of all maximal zones of the current partial view and $Data_{MZ} = data \in MZ_{set}$. If the following test is satisfied

$|Datat_{MZ}| \geqslant (1-\epsilon)|Data_{newPeer}|$, this means that effectively there is multiple clusters (so the centroid falls between them where there is no data).
To select a cluster, rather than using cost clustering algorithms, we will relay a second time on the built partial view to get possible clusters where to move. Indeed the maximal sub-zones of the partial view determine indirectly each one a cluster as we know that new peer can't get bigger than a maximal sub-zone as a responsibility zone.
Its important to notice that, even if there is data clusters in some maximal zones, in general there will be no real benefit to move because the new peer can select only one data cluster and will loose the others (it means that it will loose very probably more than the half of its data). So the new peer after selecting a cluster (or a maximal zone), will need to have a way to decide to move to it or not depending on the *possible* gain of data quantity it can keeps.

In short to deal with the case of skewed data with multiple clusters we need to answer two questions:

- How to choose a cluster (or a maximal zone)?
- How to decide if its interesting to move to the selected cluster or not?

How to choose a cluster? So the new peer need to move to a cluster that it expects it will find a sub-zone that contains more data than the actual local exact sub-zones. Choosing a cluster is possible only by choosing a maximal zone of the actual (built) partial view. The problem when choosing a cluster (i.e a Max sub-Zone) is that there will be no warranty that the whole sub-zone is a real sub-zone and is managed by one peer or it has been partitioned, because the maximal zones of the built partial view gives to the new peer just an upper-bound of the zone it can take.

There is many ways to choose a max zone, that may imply lookups to get 'ideas' about how the maximal zones are organized. And it's out of the scope of this report to discuss them as it needs a lot of details. For our tests, we have just used the naive solution which selects the maximal zone with the maximum quantity of data, to see how the heuristic performs when implying only a low computation cost. In the following we will abstract the choice of the cluster (max zone) by the function `choose_cluster(`MZ_{set},$Data_{NewPeer}$`)`

How to decide if the selected cluster is interesting or not? Once the function `choose_cluster` returns a max zone MZ_k, we still need to know if the expected gain when moving to it is interesting or not. For that we need to define at which level the expected gain is considered negligible. Also this will avoid the heuristic to loop.
We use a new threshold ϵ_{stop} which represents the minimal percentage of new peer data that it want to keep. And if the expected gain of data transfers ($\frac{|Data_{MZ_k}|}{|Data_{NewPeer}|}$) is less than ϵ_{stop}, the new peer will just select the exact sub-zone of the current partial view that contains the maximum of its data as responsibility zone and the heuristic exits.
If its not the case, this imply that the new peer still be interested by the *expected* (its important to highlight that) gain. So the new peer will run a new heuristic round for the data located in the selected cluster MZ_k.
Note: its important to notice that ϵ_{stop} threshold can be easily calculated by the new peer at *run-time*, for e.g., based on the actual available bandwidth to get how much time it will save comparing to time needed to execute a new round of the heuristic.

5.4 Experimental Evaluation

In this section we will evaluate our heuristic by building DHTs while peers join the overlay with their data. Basically the heuristic is developed to consider skewed data distributions so that there will be a possibility for each peer to find a zone to be inserted where it will keep a significant amount of its data. So we need to detect for each peer its clusters of data so that it will select the responsibility zone near to the best cluster. To do so data clustering algorithm exist and can be used by the new peer on its data. However those algorithms are too cost and can only calculate clusters and they don't guarantee that real zones containing those clusters exist. So the idea of the heuristic is to relay on the build local view to get *operational* clusters, i.e clusters that can be selected by the peer, and not calculate clusters and then find that no one can be kept by the peer due to overlay configuration. So the heuristic will avoid considerable computations.

Our typical use case is recovering a DHT where peers have crashed and data was saved in reliable storage. Our results shows that for those cases building the overlay using the heuristic to insert new peers *reduces data transfers by up to* 92% comparing to insertion using canonical algorithm. For other data distributions where the choice of zone doesn't have a considerable impacts, e.g. where new peer data is well spread over the virtual space and all zones will approximatively contain the same amount of data, our heuristic detects those cases and behave as the canonical algorithm with no overhead on data transfers. For some special cases where there is concurrency between peers, e.g. when all peers data is clustered but located in the same area of the virtual space, the heuristic may have a very low overhead (less than 2% in the worst cases we have tested) but considerably will lead to a DHT that is more operational and efficient as we will see later when discussing results.

5.4.1 Experiments setup

To join the DHT each new peer first gets an entry peer from the directory that maintains a list of peers participating in the DHT. Each peer arrives with its input data counting on overage 200 items and the size of each one between 5 and 50 Mega bytes. All tests were done using the SimGrid[14] simulator. As for the skip list, we implement the heuristic on top of the CAN DHT using the GRAS API. Note that the simulation we developed is modular and the peer's insertion algorithm can be selected at runtime, as we have now three join algorithms (canonical, using the skip list, the heuristic).

5.4.2 Results & Discussion

Skewed Data distribution - Figure 5.8 shows the results of multiple tests where peers join the DHT with very skewed data (i.e. forming one cluster), for building DHTs of different number of peers (200, 500 and 1000). The two diagrams show the difference of data transfers when using our heuristic and when using the canonical algorithm.
The first diagram 5.8.a correspond to scenario of recovering a previous DHT, i.e each peer comes with data that it managed before crash or exit. In such case, our heuristic save more than 92% data transfers than the canonical algorithm. Note that even if, each peer data was located in one zone, and no two peers clusters overlap, some transfers can occur. In fact this is related to peers join order and zone partitioning algorithm, where if one peer comes later or earlier than the first time, the partitioning dimension risk to pass over its data (whether, as a new peer or as a join peer).
The second diagram 5.8.b shows also results of tests with clustered data, where the centers of peers clusters are randomly distributed over the virtual space (following a uniform distribution). For this case

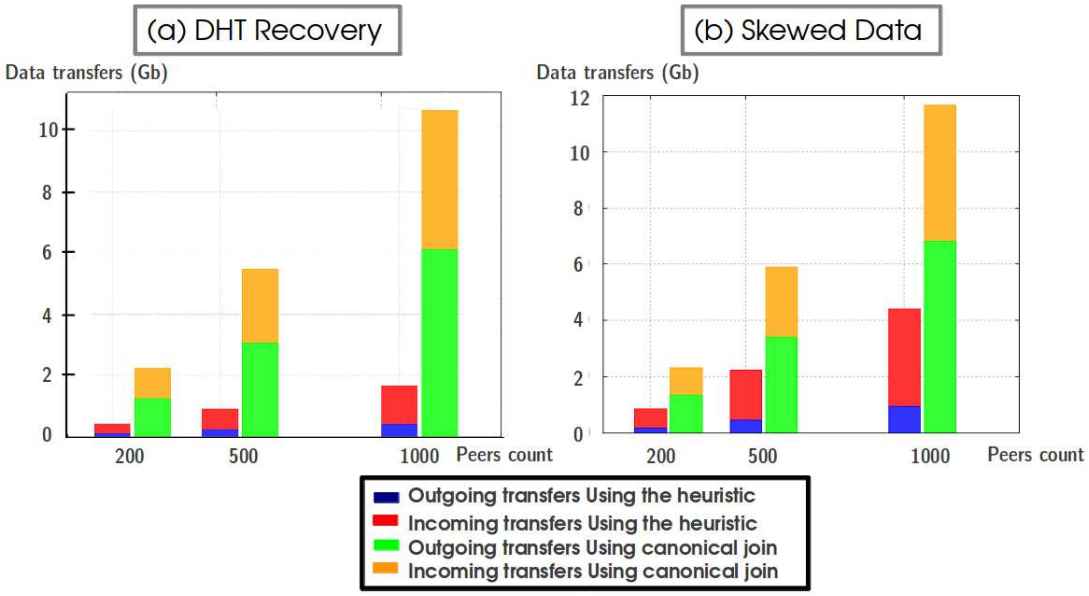

Figure 5.8: Comparison of data transfers when using the heuristic and the canonical algorithm. (a)Recovering a previous DHT (b) Building a DHT where each peer have a skewed data

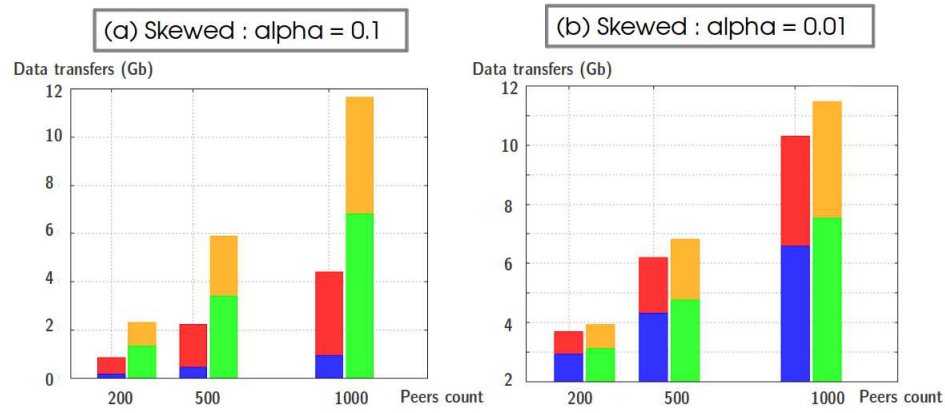

Figure 5.9: The impact of the cluster size on data transfers. Each peer have a skewed data.

the gain of using the heuristic is lower than the first scenario (65% for these tests). Indeed this is due to two main reasons that have an important impact on data transfers :

- peers data clusters may *overlap*, which will create inevitable data transfers,
- the *size* of the cluster relative to the mean size of the responsibility zone each peer will get, where if it's very big make keeping all data of new peer impossible, or possible, but later, a part of its data will be transfered to the new peer as incoming data transfers, when this peer is serving as a join peer,

To detail more the impact of the size of the cluster, we did more tests with different sizes and the results are shown in figure 5.9. We will define α as an indicator of the size of clusters:

$$\alpha = \frac{responsibility\ zone\ size}{data\ cluster\ size} \text{ and } responsibility\ zone\ size = \frac{virtual\ space\ size}{total\ number\ of\ peers}$$

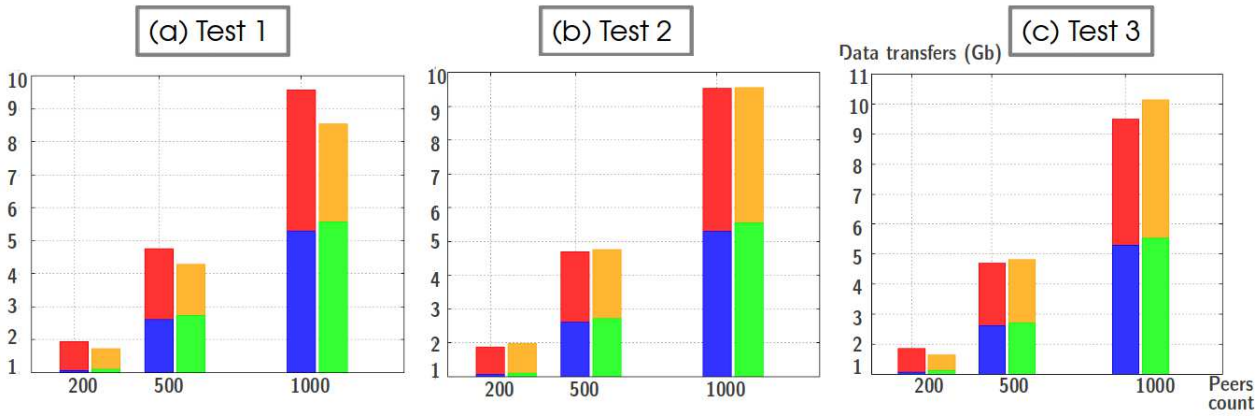

Figure 5.10: The impact of peers data cluster overlapping on data transfers. The three tests was done for the worst case where each peer data cluster overlap with all other peers clusters

Having $\alpha = 1$ means that the size of the clusters is equal to the area of the virtual space divided by the number of peers, and $\alpha = 0.5$ means that the mean of the responsibility zone that each peer will manage is the half of data cluster it have as input. For The diagram 5.9.a with $\alpha = 0.1$ the gain made by the heuristic is 65% while the second diagram 5.9.b with $\alpha = 0.01$ we have 15% of gain.

Uniform data distributions - So as the size of the clusters increase as the possible gain decrease. If we keep cluster size increasing until reaching the size of the virtual space (i.e. $\alpha = 1/total\ number\ of\ peers$) we will have peers that comes with data uniformly distributed over the virtual space. As we have seen for that case, there will be no considerable gain. Our heuristic detect those distributions and behave simply as the canonical join algorithm. For our tests, we partitioned the virtual space into 100 chunks, and we fixed $\epsilon_{CV} = 0.6$.

Overlapping data clusters - Having seen in details the impact of the size of the clusters on data transfers, lets move to the second factor: peer clusters overlapping. And lets take the worst case where each peer have clustered data but clusters are located in very small area of the virtual space making technically each peer cluster overlap with any other cluster of other peers. Figure 5.10 shows that even for those extreme cases the heuristic can be more efficient or at least will have only a minor overhead (at most 2% more transfers).
First it's important to notice that there is always a gain of outgoing data transfers and the drop of heuristic performances is due to incoming data transfers. In fact, when using the heuristic all peers will get a responsibility zone in the area where their data is located making the responsibility zones smaller an smaller. So each peer will keep only a small data, but also will receive a small quantity of data. Otherwise, when using the canonical algorithm peers will choose responsibility zones randomly in the virtual space. This will make the responsibility zones bigger but most of peers will loose all their data. Even if in some cases the heuristic can generate more data, the results still very interesting considering that :

- There may be a low overhead, knowing that no peer have *any knowledge* about other peer data.
- We still have a gain for outgoing data transfers, but the gain will be loosed when considering also incoming data transfers. This is not completely true, as the time to transfers outgoing data is *much more costly* than incoming data transfers because outgoing data need to be first routed one by one to the responsible peer and then sent, while incoming data will be transfered directly to the new peer.

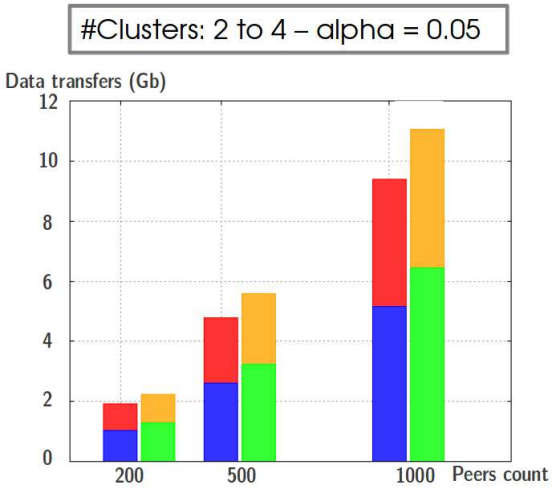

Figure 5.11: Comparison of data transfers where each peer have a skewed data but forming multiple clusters

- The resulted DHT using our heuristic is more *operational* as the DHT data load will be spread over all peers, while when using the canonical algorithm a very small number of peers will manage practically all the load, and this will imply running load balancing algorithms that mandatory generate data transfers.

Data with multiple clusters - A very challenging situation is when having peers that join the overlay with data forming multiple clusters. To minimize data transfers we need to find the best available sub-zone that contains the bigger cluster. The heuristic, first, to detect that case use a test based on simple calculation and uses a threshold that is very easy to fix a value, as we have explained in section. After many algorithms can be used to select the better cluster (or a max zone) following a compromise between precision and execution time.
Figure 5.11 shows the result of tests that use the naive method to select the best cluster (the max zone where more data is mapped). Its normal that the data transfer gain is less when having multiple clusters as the peer can keep only one cluster. As in the previous case (data forming one cluster), data transfers depends on the size of the cluster and clusters overlapping, and in addition on the number of cluster that a peer have. For example the graph shows the results of a test with a gain of 10% in data transfers, where we have $\alpha = 0.05$, data clusters number between 2 and 4, and all clusters have the same size. In our tests we used $\epsilon = 0.7$ and $\epsilon_{stop} = 0.1$.

6

Conclusion

This work has focused on building efficient DHTs. The original motivation of the work was to fastly build DHTs by minimizing uncontrollable data transfers. In the context of the *Play* project, OASIS team made use of a DHT to manage the data of a Publish/Subscribe system. The data distribution was not uniform and this made the construction of the DHT take a lot of time because of data transfers, where new peers comes with very skewed data, and if they are not inserted in the right position will transfer a lot of their data.

However, considering only the time to build the DHT, will have a bad impact on the DHT performance especially load-imbalance between peers. So this made us to think about a model that permit to regroup all interesting criteria to fastly build efficient DHTs. This model was presented in chapter 3. It is general, simple and easy-to-use. The model classifies all possible criteria in two types; *intra-DHT* and *extra-DHT* criteria; based on their dependency to the DHT and the new peer. Intra-DHT criteria depends only on the DHT status and can be calculated independently from the new peers. However extra-DHTs criteria depends also on the new peer, and ccan not be evaluated until the new peers arrives.

To develop a practical solution to consider intra-DHT criteria, we made use of a skip list: a distributed structure that lets new peers inserted in the optimal position considering multiple intra-DHT criteria. As an example, this our solution was used to ensure both load- and traffic-balancing.

In the other side, considering extra-DHT criteria is more difficult as their evaluation is different for each peer, and no computation can be anticipated before the new peer arrives. So we need a global knowledge of the DHT to let the new peer decides itself on the best position where to be inserted. Nevertheless, DHTs don't provide this global view and each peer has only a local view that include itself and a small number of its neighbors. To overcome that, we have provided an algorithm that build a partial view of the DHT more interesting than the local one. Using the partial view, we proposed a heuristic that minimizes data transfers when inserting new peers.

As a future work, considering extra-DHT criteria in the general case still an open problem. And as we have seen only a few of such criteria have been studied, and most of the proposed solutions are not distributed and/or are problem-specific. Another challenge that comes after is to mix the two solutions for intra- and extra-DHT criteria to provide a generic way for building efficient DHTs.

Bibliography

[1] Balancing locality and randomness in dhts. http://repository.cmu.edu/compsci/2238/. Accessed: 05/12/2012.

[2] Construction rapide des overlays structurés : Documentation de l'implémentation de la solution. http://code.google.com/p/data-aware-can/downloads/detail?name=D51-Implementation_solution-implemDOC.pdf. Accessed: 17/05/2013.

[3] Construction rapide des overlays structurés : Etude bibliographique. http://code.google.com/p/data-aware-can/downloads/detail?name=D21-Etude_Bibliographique-BS.pdf. Accessed: 17/05/2013.

[4] Host identity payload. http://www.ietf.org/html.charters/hip-charter.html. Accessed: 30/08/2013.

[5] Play european project, event-cloud solution. http://www.play-project.eu/solutions/event-cloud. Accessed: 30/08/2013.

[6] Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Seif Haridi, and Manfred Hauswirth. The essence of p2p: a reference architecture for overlay networks. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 11–20. IEEE, 2005.

[7] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, September 2003.

[8] K. Albrecht, R. Arnold, M. Gahwiler, and R. Wattenhofer. Aggregating information in peer-to-peer systems for improved join and leave. In *Fourth International Conference on Peer-to-Peer Computing*, pages 227–234, Aug 2004.

[9] Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. Dks (n, k, f): A family of low communication, scalable and fault-tolerant infrastructures for p2p applications. In *CCGRID*, volume 3, page 344, 2003.

[10] LucOnana Alima, Ali Ghodsi, and Seif Haridi. A framework for structured peer-to-peer overlay networks. In Corrado Priami and Paola Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 223–249. Springer Berlin Heidelberg, 2005.

[11] A. Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing, 2002. (P2P 2002). Proceedings. Second International Conference on*, pages 33–40, 2002.

[12] Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. In *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '05, pages 145–154, New York, NY, USA, 2005. ACM.

[13] Cosmin Arad, Tallat M Shafaat, and Seif Haridi. Cats: Linearizability and partition tolerance in scalable and self-organizing key-value stores. 2012.

[14] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131. IEEE, 2008.

[15] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A case study in building layered dht applications. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 97–108, New York, NY, USA, 2005. ACM.

[16] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[17] Michael Conrad and Hans-Joachim Hof. A generic, self-organizing, and distributed bootstrap service for peer-to-peer networks. In David Hutchison and RandyH. Katz, editors, *Self-Organizing Systems*, volume 4725 of *Lecture Notes in Computer Science*, pages 59–72. Springer Berlin Heidelberg, 2007.

[18] C. Cramer, K. Kutzner, and T. Fuhrmann. Bootstrapping locality-aware p2p networks. In *in: The IEEE International Conference on Networks (ICON)*, pages 357–361, 2004.

[19] T. Leighton M. Levine D. Lewin D. R. Karger, E. Lehman and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. pages 65–663, New York, NY, USA, May 1997. ACM Press.

[20] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. G-store: a scalable data store for transactional multi key access in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 163–174. ACM, 2010.

[21] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 57–66. IEEE, 2005.

[22] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, volume 7, pages 205–220, 2007.

[23] ALI GHODSI. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, Royal Institute of Technology (KTH), 2006.

[24] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. 2003.

[25] Nicholas JA Harvey, Michael B Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, volume 274, 2003.

[26] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 213–222. ACM, 2002.

[27] H. V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. Baton: a balanced tree structure for peer-to-peer networks. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 661–672. VLDB Endowment, 2005.

[28] Jimmy Jernberg, Vladimir Vlassov, Ali Ghodsi, and Seif Haridi. Doh: a content delivery peer-to-peer network. In *Euro-Par 2006 Parallel Processing*, pages 1026–1039. Springer, 2006.

[29] D. Lewin. Consistent hashing and random trees: Algorithms for caching in distributed networks. master's thesis, 1998.

[30] T. Locher, S. Schmid, and R. Wattenhofer. equus: A provably robust and locality-aware peer-to-peer system. In *in: Sixth IEEE International Conference on Peer-to-Peer Computing*, Sept 2006.

[31] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[32] Rizal Mohd Nor, Mikhail Nesterenko, and Christian Scheideler. Corona: A stabilizing deterministic message-passing skip list. *Theoretical Computer Science*, 2012.

[33] Nikos Ntarmos, Theoni Pitoura, and Peter Triantafillou. Range query optimization leveraging peer heterogeneity in dht data networks. In *Databases, Information Systems, and Peer-to-Peer Computing*, pages 111–122. Springer, 2007.

[34] L. Pellegrino, F. Baude, and I. Alshabani. Towards a scalable cloud-based rdf storage offering a pub/sub query service. In *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 243–246, 2012.

[35] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, June 1990.

[36] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31(4):161–172, August 2001.

[37] Daniel Sandler, Alan Mislove, Ansley Post, and Peter Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification. In *Peer-to-Peer Systems IV*, pages 141–151. Springer, 2005.

[38] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 73–86. ACM, 2002.

[39] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[40] Juan M. Tirado, Daniel Higuero, Florin Isaila, Jesus Carretero, and Adriana Iamnitchi. Affinity p2p: A self-organizing content-based locality-aware collaborative peer-to-peer network. *Computer Networks*, 54(12):2056 – 2070, 2010. P2P Technologies for Emerging Wide-Area Collaborative Services and Applications.

[41] Xin Yan Zhang, Qian Zhang, Zhensheng Zhang, Gang Song, and Wenwu Zhu. A construction of locality-aware overlay network: moverlay and its performance. 22 Issue 1, Jan 2004.

.

ABSTRACT

This report presents algorithms for building *Distributed Hash Tables (DHT)* or *Structured Overlays*. DHTs are used as self-managing system to handle distributed data in large networks. Our algorithms ca be used to build efficient DHTs with several properties or to recover fastly the last configuaration of a DHT after crash or exit.

In DHTs, the managed data distribution can be non uniform, and if we build the DHT using canonical algorithms it will create load imbalance between participating machines (or *peers*): it will take a lot of time and will lead to a DHT with poor performances. So to ensure efficiency in the general case, we first introduce a general easy-to-use model that can be used to consider several properties, that guarantee DHT performances. Based on that model, as a practical solution, we provide a distributed structure that permit to reorganize the DHT to facilitate the join of new machines, while considering the selected properties. As an example, this structure was used to ensure both load- and traffic-balancing over DHT machines.

To fast build DHTs, we need to have a global knowledge about DHT status, as we need to consider some properties related to the new peers characterstics. Nevertheless DHTs are distributed structures, i.e. each participating peer maintains a local state of the DHT which includes itself and a small number of its neighbors. No one peer have a global view of the whole DHT. To overcome this obstacle, we provide an algorithm that build a partial view of the DHT more interesting than the local one. 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.

RÉSUMÉ EXÉCUTIF

Ce rapport présente des algorithmes pour construire des *Tables de Hachage Distribuées* (DHT) ou des *Overlays Structurés*. Les DHTs sont utilisées pour gérer des grandes quantités de données distribuées sur des grands réseaux. Nos algorithmes peuvent être utilisés pour construire des DHTs efficaces avec plusieurs propriétés, ou pour récupérer l'ancienne configuration d'une DHT après une panne.

Dans une DHT, la distribution des données peut être non uniforme, et si on construit la DHT avec l'algorithme canonique il y aura un déséquilibre de charge entre les machines participantes (appelées aussi *pairs*): la construction va prendre beaucoup de temps et va créer une DHT avec de mauvaises performances. Donc pour assurer l'efficacité dans le cas général, nous avons introduit un modèle général, facile à utiliser qui permet de considérer plusieurs critères qui assure les performances lors de la construction de la DHT. Après, en se basant sur ce modèle, nous avons proposé comme solution pratique une structure distribuée qui permet de réorganiser la DHT pour faciliter l'insertion des nouveaux pairs tout en respectant les critères choisis. Comme exemple cette structure a été utilisée pour assurer à la fois l'équilibrage de charge et l'équilibrage de trafic entre les pairs de la DHT.

Pour construire rapidement une DHT, nous avons besoin d'une vue globale de la DHT vu qu'on doit considérer des critères relatifs aux nouveaux pairs. Cependant les DHTs sont des structures distribuées: aucun pair n'a une vue globale de toute la DHT, mais juste une vue locale composée de lui-même et un petit nombre de ses voisins. Pour surmonter cet obstacle, nous avons proposé un algorithme qui permet de construire une vue partielle de la DHT plus intéressante de la vue locale. En se basant sur cette vue, nous avons développé une heuristic qui permet de minimiser les transfers de données en considérant les données en entrées de chaque pair. Ceci a réduit considérablement le temps pris pour construire une DHT opérationnelle.