Distriubted Hash Tables and Scalable Content Adressable Network (CAN)

Ines Abdelghani

22.09.2008

Contents

1 Introduction

The current and next generations of Peer-to-Peer networks are required to be able to offer its participants efficient and secure services including data storage and exchange. By investigating the topology underlying these networks ,the main issues that have to be addressed are the location where the data can be stored, the processes to be used to retrieve a given stored data, the methods adopted to minimize the costs generated by these lookup processes and the solutions set to increase the robustness of these systems against failures and changes.
To satisfy these increasing requirements for flexibility, efficiency and robustness, the structure of the Peer-to-Peer networks has been improved from a client-server structure where a single server stores information about the location of the data to a decentralized and distributed structure: **Distributed Hash Tables.**
In the context of our survey, we will investigate in section 2 the main characteristics of this important peer-to-peer structure, which was and remains subject to several research projects. In section 3, we present the first peer-to-peer network based on the concept of distributed hash tables, which is the **Scalable Content Addressable Network** referred to as **CAN.**

2 Distributed Hash Tables: DHT

In this section, we will further discuss the interesting structure of the distributed hash tables. After introducing in section 2.1 a general idea about distributed hash tables, we will present the basic concepts underlining this structure. A categorization of distributed hash tables is introduced in section 2.3. This part of the article will be concluded by pointing out the main metrics for DHTs measurement, the properties characterizing the distributed hash tables and giving some examples for famous peer-to-peer networks using this structure.

2.1 Generalities about DHTs

Distributed hash tables has presented a popular subject for research and investigation.until today It can be considered as both an old and new topic, that attracted both academic as well as industrial interests. As mentioned in [Man03], the SDDA (Scalable Distributed Data Structures) community studied extensively this structure. One of the preliminary works in this area was elaborated by Litwin, Niemat and Shneider, who presented hash tables with central components designed for small-sized clusters. The area of research has been recently extended to cover high performance hash tables over large clusters. Distributed hash tables were also introduced as a possible structure to implement for peer-to-peer networks with millions of dynamic participants.

2.2 Basic Concepts of DHTs

To get a clear understanding of distributed hash tables, highlighting the concept of hash table is necessary. Basically, a hash table is an array to store a set of items. Every item x is mapped to a hash value $h(V)$ and then stored in slot $h(V)$ in the array. The hash function is a function:

$$h : U \rightarrow \{0, 1, \ldots, m-1\}$$

that maps each possible item in U to a position in the hash table. The parameter m is the size of the hash table.
This technique cannot be applied directly to store data in peers. As explained in [May93], this is infeasible because the number of active peers changes constantly and leads to the necessity of continuously adjusting the table's indexing. Furthermore, this would require a new allocation of data to peers with each peer departure, arrival or failure, which is very inefficient. These difficulties and performance constraints related to the direct use of hash tables in the peer-to-peer newtworks represented an incentive to develop a the concept of *Distributed Hash Table (DHT)*, which became progressively a standard method in Peer-to-Peer networks. This structure is based on the following main concepts:

1. *Mapping data values to keys*:
   Data value V is mapped to a key K using a hash function as follows:

   $$h(V) = K.$$

   The hash function needs to meet a quite demanding set of properties. First, the hash function should be easy to compute in order to ensure high efficiency of the mapping process. In addition to this requirement, the hash function should be one-way, i.e. it is hard to invert, so that for any K, it is computationally infeasible to find V such as $V = h(K)$. Another property of h is that it should be collision-free i.e. for any V it is impossible to find another V' such as $h(V)) = h(V')$.
   These targeted properties of the hash function are hard to satisfy simultaneously since they may be contradictory: for example to obtain a function that is hard to invert, the degree of difficulty to compute the value of such a function will necessarily increase. This fact make designing such functions a very challenging task.

2. *Dynamic partitioning of the keys set among nodes:*
   The interval of keys is divided in different parts and each part is associated to an active peer in the network. This partitioning is dynamic and can be efficiently adjusted by any change in the set of participants:
   In case a node newly joins the network, any active node is contacted and half of its keys subset is given to the new node. The routing structure has to be updated: the to the contacted node neighboring nodes are informed about the new node and their routing information is adequately updated.
   If a node leaves the network, the keys subset is allocated to its neighbors and the stored data is moved to the new responsible nodes. The keys set partitioning can be

adapted by nodes failure. In fact, the corresponding keys subset is allocated to other active nodes but the stored data cannot be recovered. Until the updating of the keys partitioning is done, the functioning of the network can continue by using redundant routing paths and nodes.

3. *Data Storage:*
   Once the key K is calculated, the data can be stored at the location associated to the obtained key. There are two ways of storing the data. This can be done **directly**, where data values are stored directly by the node responsible for their associated keys. Another alternative is to store pointers to where the data values are actually stored.

4. *Data lookup:*
   Any node in the network can retrieve any stored data. The requesting nodes contacts a random active node. If the data is stored at a key in the subset associated to the contacted node, there is no need for routing the data request through the network structure. Otherwise the request is spread until reaching the node responsible for storing the requested data. Several routing algorithms were developed in this context. Based on the desired features of the network (minimum latency time, high security, ...) , a certain routing algorithm can be adopted. An example for this routing algorithm will be discussed in section 3.2 as we introduce the CAN design.

2.3 Categories of DHTs

Based on how the routing is performed through the network, DHTs can be classified in two categories:

1. *Deterministic DHT:*
   This type of DHTs is characterized by only two sources of uncertainty as stated in [Man03], which are the size of the network and the mapping of subset of keys to node. In fact, the size of the network is not accurately known to all participants and the mapping of the keys to nodes is not exactly even, which adds uncertainty to the structure. In this type of networks, the overlay connection is a deterministic function of current set of node IDs. The first deterministic DHT proposals are Chord, Pastry, Tapestry and CAN, which we will further discuss in section 3.

2. *Randomized DHT:*
   The main feature of DHTs in this category is the large set of possible topologies. Unlike the deterministic DHT, where the network topology is determined by the set of nodes ids alone, the topology of the randomized DHT is chosen from this set of possible structures at run-time depending upon the random choices made by all participants. Viceroy was the first randomized protocol for DHT routing [Man03]. This increased randomness has positive effects on the robustness of the system and increases the flexibility of the routing process. However, it increases the complexity of the network.

2.4 Metrics of DHT measurement

As discussed in [NW06], there are several parameters, by which a DHT is measured. One of these parameters is the cost of join and leave. Any change in the participants set should cause a minimum disruption to the service and be accommodated easily by the system. When nodes join or leave, only a small number of participants should change their state. Another factor, which can be taken into account by measuring a DHT, is the congestion. Indeed, the performance of the service should have no bottleneck. The cost of lookups routing through the system should be evenly distributed among participating servers. Among the parameters stated in [NW06], the lookup path length and the dynamic caching addressing the problem of bottleneck caused by highly popular data items.were mentioned. In fact, the forwarding path for searching a requested data should involve as few machines as possible. Moreover, the fault tolerance against servers or connections failures represents another metric to measure a DHT.

2.5 Properties of DHTs

Distributed Hash Tables were subject to a considerable amount of attention due to their attractive properties, which can be summarized as follows [BKKRM03]:

- *Self Organization:*
  In a DHT based network, the organization and maintenance of the system is distributed among the nodes. There is no need to have a central server to manage the overall data storage and retrieval. As a result, the problem of a single-point failure is removed and the fault tolerance of the network is increased.

- *High scalability:*
  Due to their decentralized structure, DHTs are highly scalable. They can be easily extended to include a large number of peers.

- *High robustness:*
  Peers departures, arrivals and failures cause a minimum disruption of the system and affect only a part of the whole network. This fact results in a high robustness of DHT based networks against changes in the set of participants. This property is called in some refrecences **consistency**.

2.6 DHT based peer-to-peer networks

There are many peer-to-peer networks based on the concept of distributed hash tables: CAN (Scalable Content Addressable Network), Chord, Tapstry, Pastry, Kademelia, P-Grid, ...

3 Scalable Content Addressable Network: CAN

In this section, we first present the historical context of the CAN network in order to better understand the link of CAN to the general evolution of peer-to-peer networks. In a second part of this section, we discuss the basic design of CAN and introduce its several improvements alternatives.

3.1 Historical Context for CAN

The idea of peer-to-peer networks was created in 1999 by the disclosure of a client software by Shawn Napster [May93]. Indeed, Napster was the first network referred to as *peer-to-peer network*. In 2000, Justin Frankel and Tom Pepper presented the Gnutella network. Both of these networks were not scalable and the research aimed to develop network designs that can fit to very large networks such as the Internet. In this context, Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Shenker introduced in 2000 the Scalable Content Addressable Network in [RFH+01]. CAN was not only applicable to peer-to-peers systems but itcan also be used for large scale storage management systems, the construction of wide-area name resolution services and some others applications [RFH+01].

3.2 CAN Design

We investigated the features underlying the design of CAN. Here we present first the basic CAN design and then discuss the different possibilities to improve it.

3.2.1 CAN Basic Design

CAN is built on the structure of distributed hash table. Its design is based on a virtual d-dimensional Cartesian coordinate space. The dimensionality of Cartesian coordinate space is a parameter to be used for design improvement as it is shown in section can:b-design. For the 2-dimensional CAN, a square area $Q = [0, 1) \times [0, 1)$ is partitioned in rectangles and squares. Each of them is allocated to a given peer, who is responsible for storing and managing all data mapped to his rectangle.
To enable the CAN to grow incrementally [RFH+01], a new node joining the network must be allocated its own zone in the coordinate space. To start the process to join the network, the new node must find and contact an active node in the CAN, which splits its allocated zone with the new node, retaining half and handing the other half to the joining node. The keys and values data of the space part newly allocated must be transferred to the joining node. In a final phase, the neighbors of the split zone must be notified with this update in the coordinate space partitioning, so that the routing information of each node are correct and conform to the actual mapping of data to peers.
An interesting issue in CAN design is the distribution of data distribution among network participants. As stated in [May93], a uniform load distribution is low probable: Let's consider a peer p in CAN with $R(p)$ its associated rectangle. Let $A(p)$ denotes the area

of the rectangle $R(p)$. The following lemma gives an upper bound to the probability that the rectangle $R(p)$ is not split after having n peers joining the network, $P_{R,n}$:
Lemma 1:

$$P_{R,n} \leq \mathsf{e}^{-nA(p)}$$

Proof:
Let $q = A(p)$.We can express the probability that $R(p)$ is not split after the insertion of a new peer P_R as follows:

$$P_R = 1 - q$$

Based on this, the probability that the rectangle $R(p)$ is not split after having n peers joining the network $P_{R,n}$ can be calculated as follows:

$$P_{R,n} = (P_R)^n = (1-q)^n$$

We have for $\forall m \succ 0$: $(1 - \frac{1}{m})^m \leq \frac{1}{\mathsf{e}}$
Using this relationship, we obtain:

$$P_{R,n} = (1-q)^n = ((1-q)^{\frac{1}{q}})^{nq} \leq \frac{1}{\mathsf{e}}^{nq} = \mathsf{e}^{-nq} = \mathsf{e}^{-nA(p)}$$

Lemma 1 is the basis for the theorem describing the greatest area a rectangle can have after the insertion of n peers:
Theorem: In CAN, after the insertion of n peers, for the probability P_A of having a rectangle $R(p)$ with area $A(p) \geq 2c \cdot \frac{ln(n)}{n}$ we have:

$$P_A \leq n^{-c}$$

A detailed proof of this theorem can be found in [May93].

The basic operations performed by CAN are the insertion, lookup and deletion of (key, value) pairs. To understand the mechanism behind the CAN operations, we consider exemplary a key data K_1 and a value data V_1. (K_1, V_1) can be the inserted, requested or deleted data pair. For all operations, the key K_1 is mapped onto a point $R_1(x, y)$ in the coordinate space using two hash uniform hash functions h_x and h_y as follows:

$$x = h_x(K_1) \text{ and } y = h_y(K_1)$$

The point R_1 is located into a zone Z_1 in the coordinate space. This zone is allocated to a certain peer P_1, who is responsible for the storage and managing of all data associated to zone Z_1 among them the (K_1, V_1) data pair. Figure **??** illustrates how the mapping of data to peers is done in CAN. If the by CAN performed operation is a lookup for (K_1, V_1) data, we can distinguish between the requesting peer P_R and the peer P_1. Two possible scenarios may take place:

1. If P_R is the same as P_1, there is no need to forward the data request through the CAN structure.

2. If P_R is different from P_1, the data request must be routed through the network until reaching the node N_1 responsible for Z_1.

The routing performed by CAN is the so-called *Greedy routing*: it is simply following the straight line path through the Cartesian space from source to destination coordinates. Every data request message includes the destination point as the wished destination address. In fact, each CAN node has a coordinate routing table listing the IP adresses and virtual coordinates of its **immediate** neighbors and it forwards the request message to its neighbor, which is the nearest one to the destination node. The CAN design defines the procedure to be adapted by node departure:the zone associated to the leaving peer is taken over by one of his neighbors. Two scenarios can be considered at this stage: if one of the neighbors zone can be merged with the departing node zone, then this is merging operation is performed and a valid single zone is created. Otherwise, the zone is handed to the neighbor with the smallest current zone, which temporarily manage both zones simultaneously.
The maintenance of CAN is ensured by the periodic update messages, which each node sends periodically to its neighbors stating its zone coordinates and a list of its neighbors and their zone coordinates. As indicated in [May93], the prolonged absence of such update messages can be considered as a signal to the failure of a given node. Once one node notices that its neighbor is no more active, it initiates immediately the takeover mechanism. Several metrics can be taken into account by choosing which node will become responsible for the departing node zone: actual zone volume,associated load, quality of connectivity, ... It can happen that, not only one neighboring node but more than half of the neighboring nodes fail. In this case, the active node cannot take over all the zones because of the generated CAN inconsistency problem, it must perform an expending ring search for any nodes residing beyond the failure region and rebuild sufficient neighbor state to initiate a takeover safely.

3.2.2 CAN Design Improvements

The basic design described previously in section 3.2.1 can be improved at several levels. In fact, with a number of design techniques cited in [RFH+01], the latency of CAN routing can be reduced, the CAN robustness in terms of routing and data availability can be improved and a load balance can be reached. These design techniques are summarized below:

1. *Increase coordinate space dimensionality:*
   By increasing the dimensionality of the Cartesian coordinate space, the average routing path length is reduced and so is the overall routing latency. Besides, the number of neighbors increases, which implies that the routing fault tolerance is improved.

2. *Maintain multiple, independent coordinate spaces (Realities):*
   This technique consists on considering for a CAN r coordinate spaces with each

node in the system being assigned r coordinate zones, one in every reality. With the replication of the contents of hash table, the data availability is significantly improved. It is not the only advantage of this measure, better routing fault tolerance and reduced overall CAN latency are also obtained by using this technique.

3. *Overloading coordinate zones:*
   Overloading coordinate zones implies having multiple nodes sharing the same zone called *peers*. An important design parameter of this technique is the maximum number of allowable peers per zone, named MAXPEERS. Detailed description of this measure is included in the paper [RFH+01]. The main gains from this design extension is the reduced path latency and the improved fault tolerance. The cost for these gains is the increased system complexity.

4. *Use of multiple hash functions:*
   To improve data availability, a single key can be mapped onto M different points in the coordinate space by using several different hash functions. The drawbacks of this technique are the increased size of the (key, value) database and the increased query traffic.

5. *Choice of CAN routing metrics:*
   Having routing metrics reflecting the underlying IP topology of the CAN such as the network-level round-trip-time RTT favors lower latency paths. As a direct consequence, the overall CAN latency is reduced.

6. *Topologically-sensitive construction of the CAN overlay network* By constructing CAN topologies congruent with the underlying IP topology, the CAN path latency can be remarkably reduced.

7. *Performing a more uniform partioning:*
   By the insertion of a new node, the active node, which zone will be split is not necessarily the first contacted node but it can be one of its neighbors, which zone has the smallest volume. This measure leads to a more balanced load partitioning among nodes.

8. *Introducing caching and replication techniques:*
   The CAN design can be improved by applying some of the caching and replication techniques commonly used for the management of hot spot in Web. Indeed, maintaining a cache of the data keys recently accessed at the CAN node level makes very popular data widely available. Moreover, the replication of frequently requested data to neighboring nodes enables a better load spreading.

4 Conclusion

In this work, we investigated the functioning and special features of the distributed hash tables (DHT). In a second part we presented the scalable Content Addressable Network

as an example for a peer-to-peer network based on the DHT concept. We discussed its basic design and summarized the possible techniques to improve the CAN performance. However, even with all these improvement measures CAN couldn't compete with other Peer-to-Peer networks such as Chord, Pastry and Tapestry, which are more efficient in terms of network grade and routing latency

References

[BKKRM03] H. Balakrishnan, M.F. Kaashoek, D. Karger, and and I. Stoica R. Morris. Looking up data in p2p systems. *Communications of the ACM*, 2003.

[Man03] Gurmeet Sigh Manku. Routing networks for distributed hash tables. *Standford University*, 2003.

[May93] Mayer. *Peer-to-Peer Netwroks*. Addison-Wesley, Reading, Massachusetts, 1993.

[NW06] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. *Proc. SPAA*, 2006.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-adressable network. *SIGCOMM*, 2001.