Information Sciences 218 (2013) 165–181

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Fixed Interval Nodes Estimation: An accurate and low cost algorithm to estimate the number of nodes in Distributed Hash Tables

Xavier Bonnaire

Departamento de Informática, Universidad Técnica Federico Santa María, Avenida España 1680, Valparaíso, Chile

ARTICLE INFO

Article history:
Received 11 February 2010
Received in revised form 25 May 2012
Accepted 3 June 2012
Available online 12 June 2012

Keywords:
Distributed Hash Tables
Size estimation
Peer to Peer

ABSTRACT

Peer to Peer (P2P) systems have shown to be a good solution to build self-organized large scale distributed information systems. Within Peer to Peer overlays, Distributed Hash Tables (DHTs) offer strong fault tolerance properties as well as efficient object look-up algorithms. Even if the essence of a P2P system is to provide to each node a restricted local view of the whole system, it is often useful to have some global knowledge of the system, such as knowing the DHT size. The DHT size is relevant, as this information is used to adapt some parameters of the system (the number of replicas of a given object, the depth for epidemic algorithms, the potential number of clients for a given service). Computing the number of nodes in a DHT is a difficult task as it requires the best possible accuracy at the lowest achievable cost in terms of messages overhead. In this paper, we propose a new algorithm to estimate the number of nodes in a DHT called Fixed Interval Nodes Estimation (FINE). Our approach is fairly accurate for the estimation, and has a very low overhead in the number of generated messages. We give a complete set of results from simulations as well as a statistical study to demonstrate the accuracy of FINE and that it does not depend either on the DHT size, nor on the hash function used to build the DHT.

1. Introduction

Peer to Peer overlays (P2P) [1,14] have shown to be a good option to build large scale distributed information systems [18], complex trusted information sharing systems [31,4], and also emerge as mobile information sharing solutions [26]. They offer strong properties like the high availability of the stored objects, fault tolerance to network partitions and nodes failures, and nearly unlimited storage space. P2P overlays are divided into two categories, the structured overlays like Pastry [25], Chord [28], Kademlia [16] or TAPESTRY [30], and the unstructured ones like the well-known GNUTELLA [24,1].

In this paper we will only focus on a special sub-category of structured overlays called Distributed Hash Tables (DHTs) that maintain a self-organized ring.

In a DHT [5,25,28], each node obtains a unique nodeID using a cryptographic hash function H [27]. The nodeID is usually computed using the cryptographic hash of the node's IP address, $nodeID = H(@IP)$. The nodeID space is then represented by the domain of H. One of the most popular cryptographic hash functions used to build DHTs are the Secure Hash Function (SHA) [7], and the Message-Digest Algorithm 5 (MD5) [23], as these functions are known to have a good global uniform distribution of the nodeIDs within the nodeID space.

A DHT is self organized, and a node only knows a very restricted number of other nodes in the system. Depending on the overlay, a node usually maintains a set of numerically closest nodes, a set of geographically closest nodes (for example in

E-mail address: Xavier.Bonnaire@inf.utfsm.cl

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.06.004

terms of the network latency), and a routing table. Given a key k, computed using the same hash function H, the routing algorithm ensures finding either the node which nodeID is numerically closest to key k for Pastry, or the node which nodeID is the immediate successor of k for Chord. The number of hops to route a message to the node corresponding to key k is in $O(log(N))$ where N is the size of the DHT in number of nodes. This makes a DHT highly scalable, and an efficient system for objects looks-up.

Computing the number of nodes in a DHT is an important problem as. knowing the size of the DHT can be used as a configuration parameter in several algorithms to determine:

- The number of replicas needed to avoid bottlenecks, or to resist to strong Distributed Denial of Services Attacks [3].
- The growing or decreasing speed of the DHT.
- The parameters for epidemic algorithms like Random Walk or Gossip [17] used to make aggregation or multicasts.
- The potential number of clients for a given service or application.

Computing the number of nodes in a DHT appears to be a simple problem, but it is in fact a difficult one. The most accurate algorithm to do this is a flooding which consists in going through all the nodes of the DHT ring with a token, until coming back to the origin node. It can be considered as the most accurate algorithm for estimating the size of a DHT, but it has a very high cost, which makes it un-scalable, and practically unusable.

The main difficulty is then to find a trade-off between the accuracy of the computation, and its cost in the number of messages. Then, we talk about an estimation of the number of nodes in the DHT, which will always be different from the real number of nodes. We give in Section 2 a formal definition of the problem, and explain all the notations we will use in this article. Various methods have already been proposed to estimate the number of nodes in a Peer to Peer system. They are presented in the Related Work section (Section 7).

In this paper, we propose a new method to estimate the number of nodes in a DHT. Our method called Fixed Interval Nodes Estimation (FINE) is able to estimate the DHT size with a high accuracy (smaller than 1%) using a fixed interval on the DHT ring, and with a low overhead. We give the description of FINE and the way to implement the algorithm for Pastry and Chord in Section 3.

The theoretical cost of FINE in terms of the number of generated messages, and the amount of additional data are presented in Sections 3.1.1 and 3.1.2. Section 4 gives some simulation results and an evaluation of the accuracy of FINE. We also study the impact of the interval size on the accuracy, as well as the relevance of the DHT size and the DHT distribution. We then focus on the influence of the hash function used to build the overlay, comparing the three well-known hash functions *SHA-1* [7], *SHA-256* [27] and *MD5* [23], and their potential local non uniform distributions.

A summary of the results and a discussion to compare the FINE performance with existing approaches is provided in Section 5. Then, in Section 6, we give a simple mechanism to manage the persistence of the estimation, in order to further reduce the cost of FINE. Finally, the conclusion gives a summary of our contributions, and two important directions for future work.

The contributions of the paper are:

- A new size estimation algorithm (FINE) for Distributed Hash Tables,
- A fairly good accuracy of the estimation (lower than 1% of error),
- A low cost of the algorithm in number of messages,
- An accuracy and cost of the algorithm that does not depend either on the DHT real size nor on the hash function used to build the DHT.

2. Notations and problem definition

2.1. Notations

We suppose that a DHT is a set of self-organized nodes in a ring such that each node has a unique nodeID which is the result of a cryptographic hash function H. The nodeID space S is then defined by the output size O of function H:

$$O_{SHA-1} = 2^{160},\ O_{SHA-256} = 2^{256},\ O_{MD5} = 2^{128}$$

The nodeID space is then $S = [0, \ldots, 2^{O_H}]$. Except if explicitly mentioned, we suppose in all the following that the H function is the Secure Hash Function *SHA-1*. Therefore, the nodeIDs space is the set $S = [0, \ldots, 2^{160}]$. We will also assume that the *SHA* function does not generate any nodeIDs collision [22], that is, for all nodes $node_i$ and $node_j$, we have:

$$node_i \neq node_j \Rightarrow SHA(node_i) \neq SHA(node_j).$$

We note $n_j = Succ(n_i)$, the successor n_j of node n_i in the nodeID space, that is the clockwise neighbor of n_i in the ring, and $n_k = Pred(n_i)$, the predecessor of n_i (counter-clockwise neighbor) in the ring.