This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2009 proceedings.

Routing fairness in Chord: Analysis and Enhancement

Rubén Cuevas
Universidad Carlos III de Madrid
Av. de la Universidad, 30
28911-Leganes (Spain)
Email: rcuevas@it.uc3m.es

Manuel Urueña
Universidad Carlos III de Madrid
Av. de la Universidad, 30
28911-Leganes (Spain)
Email: muruenya@it.uc3m.es

Albert Banchs
Universidad Carlos III de Madrid
Av. de la Universidad, 30
28911-Leganes (Spain)
Email: banchs@it.uc3m.es

***Abstract*—In Peer-to-Peer (P2P) systems where stored objects are small, routing dominates the cost of publishing and retrieving an object. In such systems, the issue of fairly balancing the routing load among all nodes becomes critical. In this paper we address this issue for Chord-based P2P systems. We first present an analytical model to evaluate the routing fairness of Chord based on the well accepted Jain's Fairness Index (FI). Our model shows that Chord performs poorly, with a FI around 0.6, mainly due to the different sizes of the zones between nodes. Following this observation, we propose a simple enhancement to the Chord finger selection algorithm with the goal of mitigating this effect. The key advantage of our proposal as compared to previous approaches is that it does not add any overhead to the basic Chord algorithm. The proposed approach is evaluated analytically showing a very substantial improvement over Chord, with a FI around 0.9. We conduct an extensive large-scale simulation study to evaluate our proposal and validate the analysis. The simulation study includes, among other aspects, churn conditions, heterogeneous nodes and Zipf-like object popularity.**

I. Introduction

Peer-to-Peer (P2P) systems have become one of the most popular applications in the Internet, mainly pushed by file sharing applications such as BitTorrent and Emule, in addition to emerging applications such as peer-to-peer SIP (P2PSIP) [1]. Indeed, nowadays P2P applications are responsible for most of the Internet traffic [2]. P2P systems are classified into unstructured P2P systems, such as Gnutella [3], and structured P2P systems or Distributed Hash Tables (DHTs), such as Chord [4], Kademlia [5] or Pastry [6]. The main advantage of DHTs is that data placement and search procedures generate less traffic. Among the DHT-based approaches, Chord is one of the most popular systems[1] and it is the focus of the present paper.

Although the issue of fairly balancing the load among the nodes of a DHT has been extensively studied in the literature [8]–[17], most of the efforts so far have been devoted to fairly distributing the load of stored data, while the issue of balancing the routing load has received much less attention. However, as it has been observed by [10], in DHT-based P2P systems where objects are small, routing dominates the bandwidth and latency costs of storing and finding an object. This is the case for most of the existing P2P indexing systems where the objects fetched from the DHT are just lists with the IP addresses of the hosts that actually store the desired content. P2PSIP is another clear example of a P2P system with small objects. As the load of such systems is dominated by routing, it becomes critical to fairly balance the routing load among nodes.

To the knowledge of the authors, the only work that has explicitly addressed the issue of routing fairness in DHTs is [18]. This proposal is specific to Pastry and its key idea is to dynamically remove from the routing tables those nodes that are responsible for the most popular objects. One disadvantage of this approach is that it incurs additional overhead and complexity due to the dynamic maintenance of the routing tables. Additionally, the proposals made to balance the load of stored data may be also applied to the routing load; however, these also incur additional overhead. In particular, [8]–[12] assign several logical nodes (called virtual servers) to physical nodes, which adds an overhead proportional to the number of virtual servers. Finally, other approaches [13]–[17] aim at modifying the node identifiers to keep them equally spaced, which adds a high overhead to update ids when nodes join and leave the ring.

In this paper we address the issue of balancing the routing load of Chord. We first analyze the routing fairness of Chord. Then, based on this analysis, we propose a simple enhancement to the finger selection mechanism that improves very substantially Chord's fairness. The main advantage of this approach over existing proposals is that it does not add any overhead to the standard Chord algorithm. The key contributions of the paper are:

- We analyze the routing fairness of Chord according to the Jain's Fairness Index [19]. Although the goal of our analysis is to evaluate fairness, the analysis also contains other findings that are valuable contributions by themselves, including the characterization of the size of the zone between nodes, the number of fingers and the number of routed messages.
- We propose an enhancement to the basic Chord finger selection mechanism that improves very substantially the fairness of the routing load without adding any extra overhead. The fairness level of the proposed mechanism is studied analytically.

[1]For instance, Chord has been recently adopted as mandatory by the IETF P2PSIP Working Group [1].

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

- We run large scale simulations, with up to 10^6 nodes, that validate our analysis as well as the performance improvement obtained with the proposed Chord enhancement.

The rest of the paper is organized as follows. In Section II we describe the standard Chord mechanism. Section III presents an analysis of the routing fairness of Chord. Based on this analysis, we identify the main cause for Chord unfairness, which motivates the enhancement that we propose. This enhancement is described and analyzed in Section IV. Section V then evaluates the performance of Chord and the proposed mechanism, both analytically and via simulation. Finally, Section VI closes the paper with some final remarks.

II. Standard Chord Description

In this section we briefly describe the standard Chord mechanism. For a more detailed description the reader is referred to [4].

A Chord P2P system is composed of nodes arranged in a logical ring (see figure 1). The position of each node in the ring is given by the node's identifier (id). Identifiers have a length of k bits which yields an identifier space $id \in [0, 2^k - 1]$. The objects stored in the ring are identified by their key, which belongs to the same identifier space as the node ids ($key \in [0, 2^k - 1]$). Nodes' identifiers and objects' keys are usually created by means of hash operations, which results in that nodes and keys are randomly placed into the Chord ring.

The responsibility for storing an object is given to the first node whose identifier is equal to or greater than the object's key. Following this, we define the *zone of responsibility* of a node as the subset of the identifier space between this node and the previous one, since the node is responsible for all the keys that fall into this zone. For instance, in the example of figure 1, the object with key $K54$ will be stored by node $N57$, and its zone of responsibility is the id-space $(53, 57]$.

In Chord, to guarantee that the ring is not broken even when multiple nodes join and leave the system, each node knows the first s nodes whose identifier is greater than the node's. These nodes are referred to as the node's *successors*. Additionally, each node maintains a table of *fingers* which point to other nodes of the ring. In particular, a node with identifier id chooses as its fingers the nodes responsible for the keys $key_i = id + 2^{i-1}\ \forall i \in [1, k]$, respectively. In order to avoid duplicating information, the successors of a node are excluded from its finger table. Figure 1 shows the successors and fingers of nodes $N0$ and $N32$. Notice that $f1 = id + 2^0$, $f2 = id + 2^1$ are not shown because they point to the same node than $f4 = id + 2^2$. Notice also that nodes $N38$ and $N41$ are successors of node $N32$ and therefore they are not included in its finger table.

Routing in Chord is based on the following key lookup algorithm. When a node receives a query looking for a given object's key, it looks into its table of fingers as well as its successors list for the nearest node to this key that has a lower or equal identifier, and forwards the query message to this node. In this way, the query message is routed through the Chord ring until it reaches a node that detects that the key is located between itself and its successor. This node then returns the IP address of the successor which is responsible for the requested object. In the example of figure 1, if node $N0$ looks for key $K54$, it would first use its finger $f32$ to query node $N32$. Then $N32$ would route the query message through its $f16$ finger, and finally node $N53$ would return a message to $N0$ with the IP address of node $N57$, which is the node responsible for key $K54$. A key feature of this lookup algorithm is that it provides a $O(log(n))$ performance, as each finger covers a zone of size 2^i and thus allows query messages to skip half of the identifier space left towards the key.

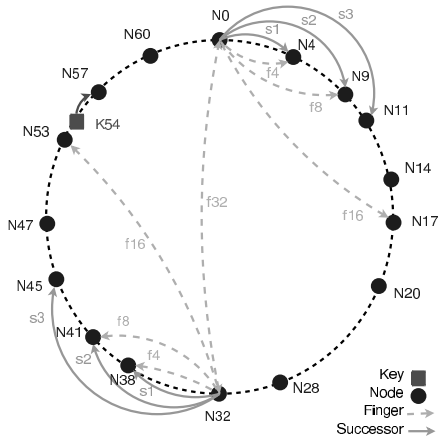

Fig. 1. An example Chord ring (k=6, s=3)

In the rest of the paper we will use the following terminology for the Chord routing operation:

- We will say that a node sends a message over a *successor link* when it forwards it to one of its successors.
- With *outbound fingers* we will refer to the outgoing fingers of a node that point to other nodes. Following our previous explanation, the successors of a node will not be considered as part of its outbound fingers.
- With *inbound fingers* of a node we will refer to the incoming fingers from other nodes that point at this node. Similarly to the above, inbound fingers will not include those nodes that have the given node as a successor.

III. Analysis of the Routing Fairness in Chord

In this section we analyze the fairness of the routing in Chord. The metric that we use to this aim is the Jain's Fairness Index (FI) [19], which is a widely accepted metric to assess the level of fairness of a distribution. This metric gives a value between 0 and 1, where 0 is the unfairest distribution and 1 is the fairest one. Applying Jain's Fairness Index to the routing load of each node in the ring (measured in terms of the number of routed messages) results in the following expression:

$$FI = \frac{|\sum_{i=1}^{n} m_i|^2}{n \sum_{i=1}^{n} m_i^2} \quad (1)$$

where n is the number of nodes in the ring and m_i is the number of messages routed by node i.

Our analysis of the routing fairness proceeds along the following steps:

- We first analyze the distribution of the size of the zone between two consecutive nodes of the Chord ring (i.e. the zone of responsibility of a node).
- Then we analyze the number of *inbound fingers* pointing to a node as a function of the size of the zone of responsibility of that node and, from this, the distribution of *inbound fingers*.
- Next, we obtain the number of messages routed by a node as a function of its number of *inbound fingers*.
- Finally, we compute the FI of the routing load based on the distribution of the number of messages routed by one node.

Without loss of generality, our model assumes that node identifiers are randomly distributed in the continuous id-space [0,1]. This is an accurate approximation of the real scenario in which the node-ids are randomly distributed in the discrete id-space [0,2^k), where k is the length of the hash operation employed to generate identifiers (e.g. 128, 160 or 256). Note that with such large values of k, the assumption that the id-space is continuous instead of discrete yields very accurate results. The conversion of the space [0,2^k) to [0,1] only introduces a scalar factor whose only purpose is to simplify the notation.

Let us start with the analysis of the size of a node's zone of responsibility in Chord. Let X be the random variable that represents the size of the zone between two consecutive nodes. The probability of the node's zone (X) to be smaller than a given value x is equal to the probability of having at least one node inside X. Since nodes ids are uniformly distributed in the ring and the id-space size is equal to 1, the probability that a node falls within a zone of size x is precisely x. This yields the following cumulative distribution function for random variable X:

$$cdf\,(x) = P\,(X \leq x) = 1 - (1-x)^n \tag{2}$$

The probability distribution function of the size of the node's zone of responsibility, $pdf(x)$, is simply computed as the derivative of the above cdf:

$$pdf(x) = n(1-x)^{n-1} \tag{3}$$

We next model the probability $P(f|x)$ that a node has f *inbound fingers* given that it has a zone of size x. Our key approximation in the modeling of $P(f|x)$ is to assume that all the fingers in the ring are uniformly distributed in the id-space. From this assumption, the probability that a node whose zone size is x has f inbound fingers can be computed as the probability that f fingers fall into a zone of size x;

$$P(f|x) = \binom{F}{f} x^f\,(1-x)^{F-f} \tag{4}$$

where F is the total number of fingers in the ring.

To compute the above expression, we need to obtain F, which we do as follows;

$$F = nf_{out} \tag{5}$$

where f_{out} is the average number of *outbound* fingers of Chord nodes.

To calculate f_{out}, we consider the following behavior of Chord. A node with a given id looks for outbound fingers in id-space $[id, id+2^k)$ as follows. It divides the id-space in k zones, namely $[id+2^i, id+2^{i+1})$, with $i = 0, 1, 2, ..., k-1$. Then, it establishes an outbound finger to one of these zones if the following two conditions hold:

- There should be at least one node in the zone $[id+2^i, id+2^{i+1})$, as otherwise no node can be selected as a finger. This occurs with probability:

$$P_{node}(i) = 1 - \left(1 - \frac{2^i}{2^k}\right)^n \tag{6}$$

- There should be at least s successors in the zone $[id, id+2^i-1]$, i.e. between the node id and the given zone, as otherwise this will be a successor link and not an outbound finger. This occurs with probability:

$$P_{succ}(i) = 1 - \sum_{j=0}^{s-1} \binom{n}{j} \left(\frac{2^i-1}{2^k}\right)^j \left(1 - \frac{2^i-1}{2^k}\right)^{n-j} \tag{7}$$

Assuming that the above conditions are independent, we can compute the average number of *outbound* fingers of a node as the sum across all the zones of the probability that the node has an outbound finger pointing to a node in the zone:

$$f_{out} = \sum_{i=0}^{k-1} P_{node}(i) P_{succ}(i) \tag{8}$$

The above terminates the analysis of $P(f|x)$. The next step is to compute the distribution of the number of inbound fingers, $P(f)$. We do this by combining Eqs. (3) and (4)[2],

$$\begin{aligned} P(f) &= \int_0^1 P\,(f|x)\,pdf\,(x)\,dx \\ &= \int_0^1 \left(\binom{F}{f} x^f\,(1-x)^{F-f}\right)\left(n\,(1-x)^{n-1}\right) dx \\ &= \binom{F}{f} n \int_0^1 x^f\,(1-x)^{F+n-f-1}\,dx \\ &= \binom{F}{f} n \frac{f!\,(F+n-f-1)!}{(F+n)!} \end{aligned} \tag{9}$$

We next address the analysis of the average number of messages routed by a node (m) as a function of the number of inbound fingers of that node (f). The key approximation here is to assume that all successor links in the ring route the same number of messages and so do all fingers. With this approximation the number of messages routed by a node can be expressed as a function of the number of *inbound* fingers

[2]To solve the integral we have used $\int_0^1 x^a(1-x)^b\delta x = \frac{a!b!}{(a+b+1)!}$.

pointing to this node as follows[3];

$$m(f) = (s-1)M_s + (f+1)M_f \tag{10}$$

where M_s is the average number of messages routed by each successor link and M_f the average number of messages routed by each finger.

In the above expression, M_s and M_f remain to be computed. We start with M_s. Let q be the total number of queries routed in the ring during a given observation interval, and H_s the average number of hops per query over successors links. Then, M_s can be computed as the total number of hops routed over successors links during the interval, qH_s, divided by the total number of successor links in the Chord ring, $n(s-1)$, which yields:

$$M_s = \frac{qH_s}{n(s-1)} \tag{11}$$

In order to compute H_s in the above expression we proceed as follows. When routed towards a certain destination, a query message may either reach the destination node directly via a finger, or it may first reach one of the $s-1$ nodes that has the final destination in its successors list. In the former case, there isn't any hop in successors links, while in the latter there is one. Assuming that any of these s nodes are reached with the same probability, the second event occurs with probability $(s-1)/s$, which yields:

$$H_s = \frac{s-1}{s} \tag{12}$$

To compute M_f we proceed similarly to Eq. (11): we divide the total number of hops routed over fingers by the total number of fingers in the ring, which gives

$$M_f = \frac{qH_f}{n(E(f)+1)} \tag{13}$$

where H_f is the average number of hops per query routed over fingers, and $E(f) = F/n$ is the average number of fingers per node.

To compute H_f we follow the result of [4] which shows that the average number of hops in a Chord ring that does not use successor links to route queries is given by $\frac{1}{2}log_2(n)$, where n is the number of nodes of the ring. In order to apply this result in a ring that uses successor links, we assume that this expression holds when considering a subportion of the ring. In particular, we consider that the number of hops using fingers required to cover a zone with s successors is given by $log_2(s)/2$. Since, by using successor links we save the hops required to cover this zone, these do not have to be counted in H_f, which gives

$$H_f = \frac{log_2(n)}{2} - \frac{log_2(s)}{2} \tag{14}$$

[3]We account for the last successor as a finger instead of a successor because of the following. The number of messages that a node routes through a link depends on the size of the zone between this link and the next one. Since successors are close to each other, this size will be small for all successors but the last one. Indeed, the link next to the last successor is a finger, and the zone inbetween is larger than the zone between two successors. Following this, in our analysis we take $s-1$ for the number of successors and $f+1$ for the number of fingers.

The above terminates the analysis of $m(f)$; combining all the equations we obtain:

$$m(f) = \frac{q}{n}\left(H_s + (f+1)\frac{H_f}{E(f)+1}\right) \tag{15}$$

In summary, our analysis has shown that a node routes a number of messages $m(f)$ given by Eq. (15) with a probability $P(f)$ given by Eq. (9), for $f = 0, 1, 2, \ldots, F$.

With the above, we can proceed to compute the Fairness Index (FI) given by Eq. (1), which is the objective that we set at the beginning of this section. The numerator of Eq. (1) corresponds to the square of the total number of messages routed. Since the total number of queries is given by q, and each query takes $H = H_s + H_f$ hops in average, we have:

$$\left|\sum_{i=1}^{n} m_i\right|^2 = (qH)^2 \tag{16}$$

To compute the denominator of Eq. (1), we rearrange the sum by grouping all the nodes that have f fingers (and therefore route $m(f)$ messages according to our previous analysis) together:

$$n\sum_{i=1}^{n} m_i^2 = n\sum_{f=0}^{F} n_f m^2(f) \tag{17}$$

where n_f is the number of nodes that have f inbound fingers. Since the probability that a node has f fingers pointing to it is given by $P(f)$, we have:

$$P(f) = \frac{n_f}{n} \tag{18}$$

from which:

$$n\sum_{i=1}^{n} m_i^2 = n^2\sum_{f=0}^{F} P(f)m^2(f) \tag{19}$$

Combining Eqs. (1), (16) and (19) we obtain:

$$FI = \frac{(qH)^2}{n^2\sum_{f=0}^{F} P(f)\left(\frac{q}{n}\left(H_s + \frac{H_f}{E(f)+1}(f+1)\right)\right)^2} \tag{20}$$

from which we finally obtain the following expression:

$$FI = \frac{H^2}{H_s^2 + \sum_{f=0}^{F} P(f)\left(\left(H_f\frac{f+1}{E(f)+1}\right)^2 + 2H_sH_f\frac{f+1}{E(f)+1}\right)} \tag{21}$$

which terminates the analysis routing fairness in Chord.

IV. Enhanced Chord: Proposal and Analysis

Following the analysis presented above, it can be observed that the routing unfairness of Chord is caused by the different size of the zones between nodes. Indeed, since ids are randomly placed in the ring, it is well known that zones will typically have very different sizes [4]. Then, the larger the size of the zone of responsibility of a node, the larger the probability that a finger falls within this zone and therefore the more inbound fingers this node will have. Since the number of

messages routed by a node grows with the number of fingers that point to the node, this yields a large degree of unfairness. In this section, we propose (and analyze) an enhancement of Chord that improves the routing fairness without changing the distribution of nodes and zones between them. We name our solution Enhanced Chord (e-Chord).

Since, according to the above, routing unfairness in Chord is caused by the unbalanced distribution of inbound fingers, the main goal of e-Chord is to modify the finger selection algorithm in order to achieve a more balanced distribution. With Chord's original finger selection algorithm, a node with a larger zone is more likely to be chosen as a finger; in order to avoid this, the key idea of e-Chord is the following. When a node is selected to be a finger, instead of pointing the finger to this node, we choose with some probability one of the node's successors and point the finger to this successor. In particular, when choosing a finger, the node and all its successors are selected with exactly the same probability. In this way, the number of inbound fingers of a node does no longer keep a strong correlation with its zone size, since a node with a large zone will share incoming fingers with its successors.

In more detail, the finger selection algorithm of e-Chord works as follows. Like in basic Chord, when a node with an identifier id creates its fingers table, it starts by performing a lookup to know which are the nodes n_i with an identifier equal to or greater than $key_i = id + 2^{i-1}\ \forall i \in [1,k]$. Then, instead of selecting this node as the finger (as it would be done in basic Chord), the finger is picked up randomly from the set of $R = s+1$ nodes formed by node n_i and its s successors.

Note that this mechanism does not require e-Chord nodes to keep any additional information than with standard Chord, since nodes only need to know their s successors. Moreover, the random process for finger selection of e-Chord can be performed at the remote node n_i in a transparent way without adding any extra overhead. In particular, when a node issues a query to set up a finger and this query reaches the destination node, instead of returning the IP address of n_i, it can simply return with some probability the address of one of its successors.

Another advantage of the proposed solution is that it does not require to modify any other Chord mechanism and in particular the Chord lookup algorithm is not modified at all. Additionally, e-Chord also keeps the $O(log(n))$ performance of the Chord lookup algorithm. Indeed, this performance is given by the fact that fingers are located in different 2^i zones of the Chord ring and does not depend on the specific nodes being used as fingers within these zones.

In summary, with a simple modification to the standard Chord finger selection mechanism, e-Chord improves the fairness of the routed messages without adding any control traffic overhead. We next analyze the resulting routing performance; the analysis follows the same lines as the analysis of the routing fairness of Chord in the previous section, although there are some significant differences:

- Instead of computing the distribution of the zone between one node and the next one, we now compute the distribution of the zone that contains a node and its s successors. We refer to this as a *R-node zone*, where $R = s+1$ is the number of nodes contained in this zone.
- The probability that a node is chosen as a finger is computed as $1/R$ times the probability that it falls into the node's R-zone.
- Once the number of inbound fingers that point to a node has been computed, the rest of the analysis proceeds like for the Chord case.

To analyze the distribution of an R-node zone we proceed as follows. Let its size be represented by the random variable X. Then, the probability that this variable is smaller than a give value x is equal to the probability that more than R nodes have their identifiers within this zone. Since the node ids are uniformly distributed along the $[0,1]$ id-space, the probability that one given id falls in this area is x, which yields:

$$cdf\,(x) = Pr\,(X < x) = 1 - \sum_{r=0}^{R-1} \binom{n}{r} x^r\,(1-x)^{(n-r)} \quad (22)$$

The pdf of the R-node zone is computed as the derivative of the above:

$$pdf\,(x) = \sum_{r=0}^{R-1} \binom{n}{r} (nx - r)\,x^{(r-1)}\,(1-x)^{(n-r-1)} \quad (23)$$

For a finger to point to a given node whose R-node size is x, the following two things must happen: i) the finger key has to fall within the node's R-zone, which occurs with probability x, and ii) the node must be selected among the R possible candidates for this finger, which occurs with probability $1/R$. This yields a probability of x/R for a node to be chosen as a finger. Following this, we can compute the probability that a node has f *inbound* fingers as follows:

$$P(f|x) = \binom{F}{f} \left(\frac{x}{R}\right)^f \left(1 - \frac{x}{R}\right)^{(F-f)} \quad (24)$$

The next step is to compute the distribution of the number of inbound fingers in e-Chord, $P(f)$:

$$\begin{aligned} P(f) &= \int_0^1 P(f|x) pdf(x) dx \\ &= \sum_{r=0}^{R-1} \binom{n}{r} \binom{F}{f} \frac{1}{R^f} \int_0^1 (nx - r)\,x^{r+f-1} \\ &\quad (1-x)^{n-r-1} \left(1 - \frac{x}{R}\right)^{F-f} dx \end{aligned} \quad (25)$$

By applying the following two equalities to the above,

$$(1-x)^z = \sum_{k=0}^{z} \binom{z}{k} 1^k\,(-x)^{z-k} \quad (26)$$

$$\left(1 - \frac{x}{R}\right)^z = \sum_{k=0}^{z} \binom{z}{k} 1^k \left(\frac{-x}{R}\right)^{z-k} \quad (27)$$

we obtain:

$$
\begin{aligned}
P(f) = & \sum_{r=0}^{R-1} \sum_{k=0}^{n-r-1} \sum_{j=0}^{F-f} \binom{n}{r} \binom{F}{f} \binom{n-r-1}{k} \\
& \binom{F-f}{j} \frac{1}{R^{F-j}} (-1)^{n-r-1-k+F-f-j} \\
& \int_0^1 (nx - r)\, x^{F+n-k-j-2} \delta x) \\
= & \sum_{r=0}^{R-1} \sum_{k=0}^{n-r-1} \sum_{j=0}^{F-f} \binom{n}{r} \binom{F}{f} \binom{n-r-1}{k} \\
& \binom{F-f}{j} \frac{1}{R^{F-j}} (-1)^{n-r-1-k+F-f-j} \\
& \left(\frac{n}{n+F-k-j} - \frac{r}{n+F-k-j-1} \right)
\end{aligned}
\tag{28}
$$

Once $P(f)$ has been obtained, the rest of the analysis to compute the Fairness Index of e-Chord routing follows the same Eqs. (10) – (21) of the basic Chord analysis. This terminates the analysis of the routing fairness in e-Chord.

V. Performance Evaluation

In this section we evaluate the performance of the basic Chord mechanism and the proposed enhancement in terms of routing load and fairness. Furthermore, we validate the presented analytical model by comparing it against simulations results. Simulations have been performed using our own simulator developed in Java[4]. For all simulations, 95% confidence intervals are given with errorbars (note that in many cases they are so small that they can barely be appreciated in the graphs). Unless otherwise stated, we take a number of nodes $n = 10^6$ and a number of successors $s = 16$ as default values. Note that these are typical settings in real Chord implementations [20] and real P2P applications [7]. The number of queries in each simulation run is $q = 10^8$, and for each query message a pair of source and destination nodes is chosen randomly among all nodes (except for the experiment of Section V-G in which we consider a Zipf-like object popularity distribution).

A. Routing fairness

We first evaluate the routing fairness of Chord and e-Chord. For this purpose, we use the Jain's Fairness Index (FI) applied to the routing load as proposed in Eq. (1). Table I gives the FI obtained via analysis and simulation for different settings, in the ranges $n \in [10^3, 10^6]$ and $s \in [8, 32]$. We draw the following conclusions from these results:

- Simulation follow analytical results fairly closely, with small errors of around 1% and 2% both for Chord and e-Chord. This validates the analyses presented in Sections III and IV.
- The proposed e-Chord scheme outperforms Chord very substantially. Indeed, the fairness index provided by Chord is around 0.6 for all n and s settings, while e-Chord provides a much greater fairness index, around 0.9. This validates the proposed approach as it provides a much better routing fairness than standard Chord.

[4]The source code of the simulator is available under a GPL license at http://www.it.uc3m.es/muruenya/infocom09/ChordSim.jar.

n	s	chord simulation	chord analysis	e-chord simulation	e-chord analysis
10^3	16	0.6470	0.6726	0.9029	0.9109
10^4	16	0.6024	0.6166	0.8996	0.9126
10^5	16	0.5752	0.5882	0.9039	0.9163
10^6	16	0.5594	0.5710	0.9064	0.9198
10^6	8	0.5596	0.5638	0.8816	0.8886
10^6	24	0.5591	0.5746	0.9149	0.9309
10^6	32	0.5618	0.5772	0.9189	0.9360

TABLE I
Fairness Index of the messages routed by Chord/e-Chord nodes

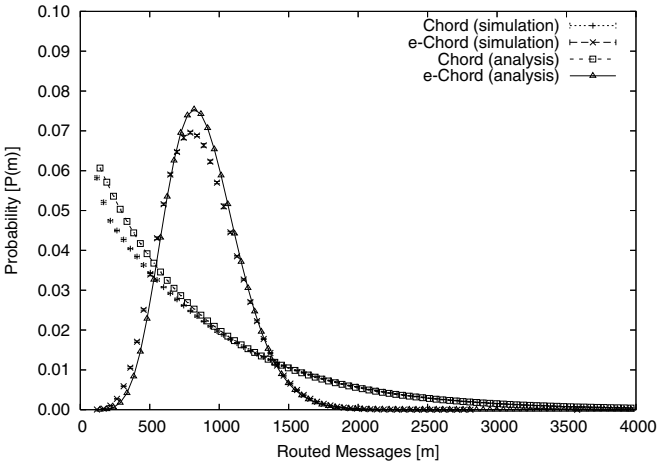

Fig. 2. Histogram of the messages routed by Chord/e-Chord nodes ($n = 10^6, s = 16, q = 10^8$)

B. Routing load distribution

The results given in the previous section allow to assess the overall level of fairness of the routing load but do not show how the routing load is distributed among nodes. In order to give an insight into the actual distribution of the routing load, figure 2 illustrates the distribution of the number of routed messages (namely, the probability with which a node routes a given number of messages). The results are obtained both analytically and via simulation.

We can observe from the figure that the routing load with e-Chord is much more balanced that with Chord, which explains the FI results obtained in Section V-A. Specifically, with e-Chord the distribution is centered around the average and sharply decreases for smaller and larger numbers, which means that most nodes route a number of messages close to the average. Instead, with Chord we have a much less balanced distribution, with with many nodes that route few messages and a long tail with heavily loaded nodes.

We further observe that the analytical results follow simulations reasonably closely, although there is some small deviation. This deviation is caused by our assumption that all

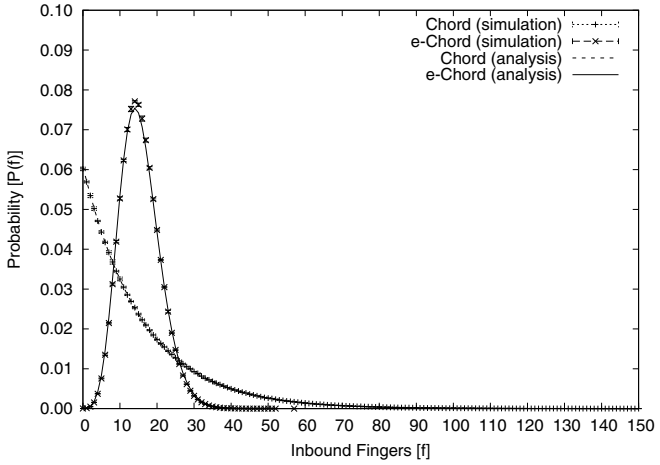

Fig. 3. Distribution of inbound fingers pointing to Chord/e-Chord Nodes ($n = 10^6, s = 16$)

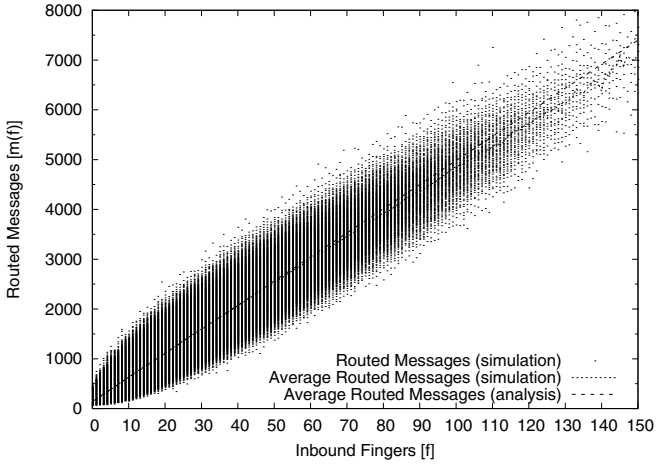

Fig. 4. Routed messages vs. inbound fingers per node in Chord ($n = 10^6, s = 16, q = 10^8$)

the nodes that have the same number of fingers route the same number of messages. This assumption hides some randomness in the number of routed messages that our model does not account for.

C. Inbound fingers distribution

The key idea behind e-Chord is to balance the number of fingers that point at each node by redistributing the inbound fingers that a node receives among its successors. In order to evaluate how well e-Chord achieves this objective, figure 3 depicts the distribution of the number of fingers for Chord and e-Chord (in particular, the figure shows the probability that a node has a given number of inbound fingers f), obtained analytically and via simulation.

We first observe from the figure that e-Chord provides a much more balanced distribution than Chord. Indeed, with e-Chord almost all nodes have a number of fingers close to the average given by the peak in the distribution, while Chord exhibits a much more unbalanced distribution. Note that distributions have a similar shape to the ones of figure 2.

We conclude that e-Chord achieves the objective of balancing the number of inbound fingers. We further observe from the figure that analytical results follow simulations very closely, which validates this part of the analysis.

D. Routing load vs. number of fingers

One key step of our analysis is the computation of the number of messages routed by a node as a function of the number of fingers pointing to this node. This relationship is given by Eq. (15). In order to validate this part of the analysis, we performed the following experiment. We evaluated the number of messages routed by all the nodes that have the same number of fingers, took their average and compared it against the value given by Eq. (15).

Figure 4 illustrates the number of messages routed by all nodes, their mean as well as the value given by the analysis. We observe that the average value matches with the analytical results, which validates this part of the analysis. There are only some deviations for large numbers of fingers, however this region contains only few samples which make the results not statistically significant.

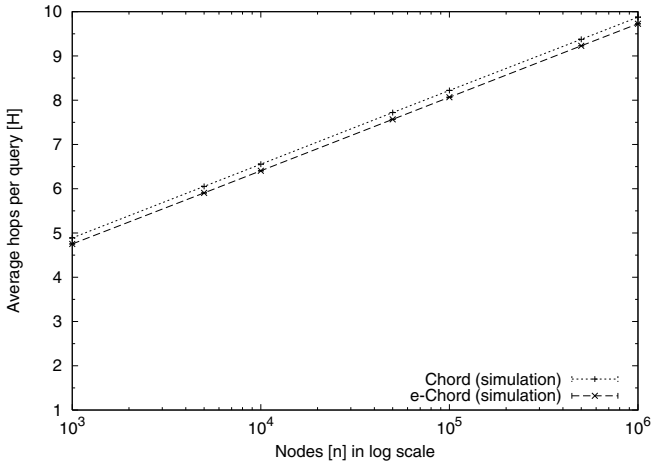

Fig. 5. Average number of hops per query in Chord/e-Chord ($s = 16$)

E. Number of hops

Since e-Chord introduces some changes in the set up of the fingers, this could raise the question of whether choosing different fingers can have a negative impact on routing. In order to evaluate this, we measured the routing performance in terms of average number of hops for rings of different size n. The results obtained are illustrated in figure 5. We observe from these results that routing in e-Chord is not negatively affected by the different choice of fingers, and not only that, but it actually performs slightly better than Chord since fingers are better spread over the e-Chord ring. Therefore we conclude that e-Chord does not pay any price in lookup performance to achieve a better routing fairness.

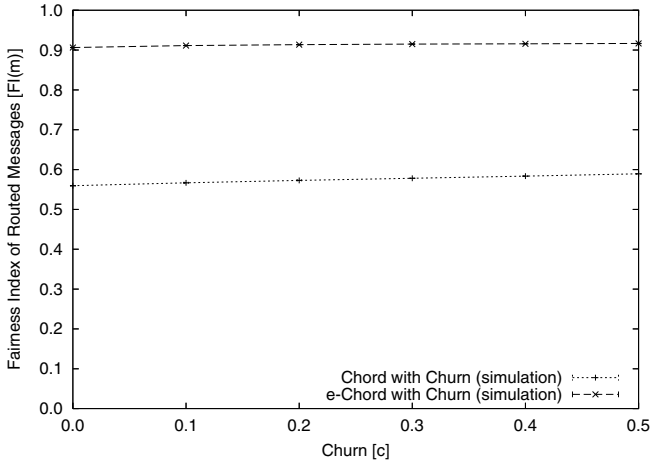

Fig. 6. Fairness index of the messages routed by Chord/e-Chord nodes under Churn ($n = 10^6, s = 16, q = 10^8$)

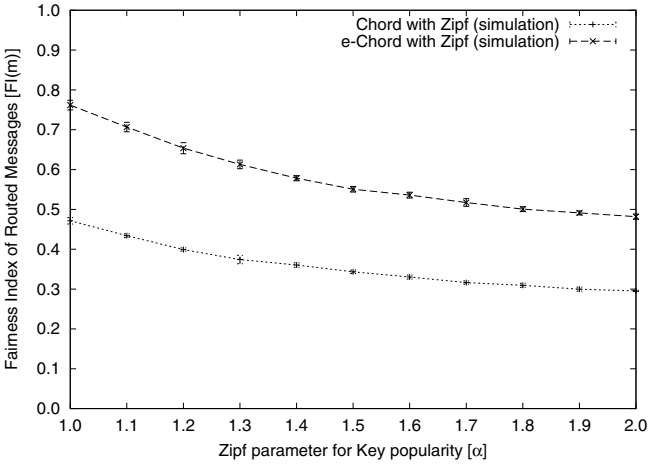

Fig. 7. Fairness Index of the messages routed by Chord/e-Chord nodes with $Zipf(10^5, \alpha)$ object popularity ($n = 10^6, s = 16, q = 10^8$)

F. Churn conditions

Our analysis and simulations so far have assumed static conditions in which all nodes are permanently active. In order to gain insight into the impact of churn conditions into the performance of Chord and e-Chord, we ran the following experiment. We divided the simulation into cycles of 10.000 queries each. At the beginning of each cycle, each node leaft the system with a probability $c \in [0.0, 0.5]$ to return in the next cycle. The fairness indexs FI resulting from this experiment with Chord and e-Chord are given in figure 6. We observe that the FI of Chord and e-Chord keeps almost constant independent of c; there is only a slight improvement when c grows which is caused by the fact that the worst-off nodes see their load reduced as a result of being inactive for some time. In any case, e-Chord clearly outperforms Chord regardless of c, which confirms the effectiveness of the proposed solution also under churn conditions.

G. Zipf popularity

All the simulations so far has assumed that all objects are equally popular. While this assumption may be appropriate for some scenarios like P2PSIP, it may be less suitable for other scenarios such as file sharing. Indeed, Zipf have been shown to be a good approximation [21] of the popularity distribution in P2P systems and have been widely used [18] to reflect the skewed popularity level of the different contents in a file sharing scenario.

In order to understand the performance of Chord and e-Chord under a Zipf popularity distribution, we simulated them and evaluated the resulting FI varying the α parameter of a Zipf distribution with the most popular key being queried from $q_{max} = 10^5$ different nodes. The results, given in figure 7, show that both Chord and e-Chord become unfairer as the Zipf α parameter increases. The reason is that, when some objects become a lot more popular than others, the routes that lead to the popular objects become more loaded independent of the algorithm used. However, e-Chord keeps outperforming Chord by a similar degree, which confirms the effectiveness of e-Chord also for this case.

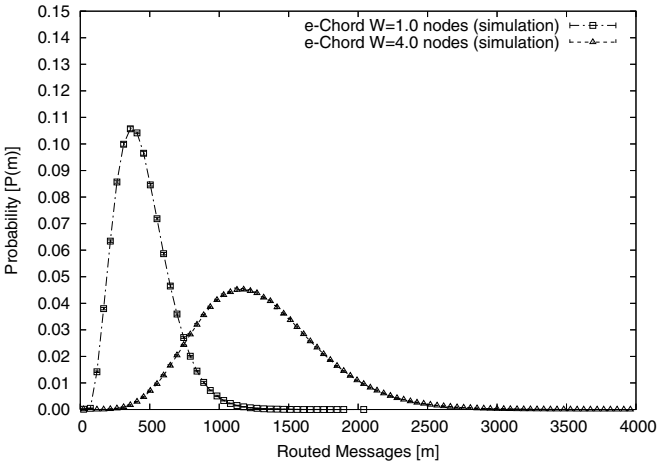

Fig. 8. Distribution of the messages routed by heterogeneous Chord/e-Chord Nodes ($n = 10^6, s = 16, q = 10^8$)

H. Heterogeneous nodes

In case of heterogeneous nodes, it may be desirable to assign more load to those nodes that have a larger capacity and/or computational power. While other approaches [8]–[12] require of additional overhead to support this functionality, with e-Chord this can be achieved just by i) assigning different weights to the different nodes depending on their capacity, and ii) accounting for these weights when redirecting fingers among successors. In order to assess the feasibility of e-Chord with an heterogeneous population, we performed the following experiment. We divided the nodes of the ring in two categories: low capacity and high capacity nodes, and assigned a weight $W = 1$ to the former and $W = 4$ to the latter. Figure 8 depicts the distribution of the routing load for both categories. It can be observed that the load of the high capacity nodes is larger than the load of the low capacity nodes, which confirms the

ability of e-Chord to support heterogeneous scenarios without incurring any extra overhead.

VI. Summary and conclusions

While fairness in P2P has been widely studied for stored data, much less effort has been devoted to studying the fairness on routing load. In this paper we have argued that routing represents the main cost in those DHT-based P2P systems where the size of the stored objects is small, and we have addressed this issue for Chord.

Our analysis of the routing fairness in Chord has been based on applying the well accepted Jain's Fairness Index to the routing load supported by each node in the Chord ring. To compute this metric, we have modeled the distribution of the routing load. We have done this by $i)$ computing the distribution of the size of the zone between two consecutive nodes, $ii)$ calculating from this the distribution of the number of fingers that point to a node, and $iii)$ obtaining from this the routing load distribution. The analysis we have conducted has shown that Chord's routing unfairness is mainly caused by the different zone sizes between nodes. Indeed, the larger the zone of responsibility of a node, the more fingers it will receive and therefore the larger its routing load will be.

In order to mitigate Chord's unfairness, in this paper we have proposed a simple enhancement to the finger selection mechanism of Chord. In the proposed algorithm, when receiving a query to set up a finger, a node replies with the address of one of its successors chosen randomly. This balances the number of fingers that point to a node, since this number does no longer depend on the size of its zone, which results in a more balanced routing load. One key advantage of the proposed approach is that, since a node in Chord already knows the addresses of all its successors, the proposed algorithm does not add any extra overhead.

The analytical results obtained in this paper, which have been validated by simulation, show that the proposed enhancement to Chord improves very substantially the performance of Chord. Indeed, the Jain's Fairness Index, which provides a measure between 0 and 1, is about 0.6 for the number of messages routed by Chord nodes, while the enhanced Chord proposal achieves a fairness index of 0.9. In addition to the fairness index, we have also evaluated analytically the distribution of the number of fingers and the routing load. The results obtained show that enhanced Chord provides a much better balanced distributions, which explains the big difference in fairness indexes.

In addition to the analysis, we have also conducted a simulation study to gain insight into the performance of the proposed solutions. Simulations have been performed for up to 10^6 nodes, which corresponds to realistic implementations of P2P systems. Simulation results have shown that the improvement in fairness of our solution does not involve any price in performance; on the contrary, it behaves even slightly better than Chord in terms of the number of hops required to reach the destination of a query. Simulation results have also shown the effectiveness of the proposed solution under churn conditions, Zipf-like object popularity and heterogeneous nodes.

Acknowledgements

This work has been partially supported by the EU funded CONTENT NoE (FP6-IST-038423, http://www.ist-content.eu).

References

[1] C. Jennings, B. Lowekamp, E. Rescorla, S. A. Baset and H. Schulzrinne. "REsource LOcation And Discovery (RELOAD) <draft-ietf-p2psip-reload-00.txt>". 2009.

[2] "Ipoque Internet Study 2007 <http://www.ipoque.com/userfiles/file/internet_study_2007.pdf>". September 2007.

[3] T. Klinberg, R. Manfredi, "Gnutella 0.6 <http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.txt>". June 2002.

[4] I. Stoica, R. Morris, D. Karger, M.F Kaashoek, H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for internet applications". Proc. of ACM SIGCOMM, 2001.

[5] P. Maymounkov, D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric". Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[6] A. Rowstron and P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale Peer-to-peer systems". Proc. of 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), 2001.

[7] M. Steiner, T. En-Najjary, and E.W. Biersack, "A global view of KAD". Proc. of 7th SIGCOMM conference on Internet Meassurements (IMC), 2007.

[8] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems". Proc. of IEEE Infocom, 2004.

[9] A. R. Karthik, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured P2P systems". Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS), 2003

[10] B. Godfrey and I. Stoica, "Heterogeneity and load balance in distributedhash tables". Proc. of IEEE Infocom, 2005.

[11] Y. Zhu and Y. Hu, "Towards efficient load balancing in structured P2P systems". Proc. of 18th International Parallel and Distributed Processing Symposium (IPDPS), 2004.

[12] Y. Zhu and Y. Hu., "Efficient, proximity-aware load balancing for DHT based P2P systems". IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 4, pp. 349-361, 2005.

[13] J. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables". Proc. of 2nd International Workshop on Peer-to-Peer Systems (IPTPS), 2003.

[14] K. Kenthapadi and G. S. Manku, "Decentralized algorithms using both local and random probes for P2P load balancing". Proc. of 17th ACM Symposium on Parallelism in Algorithms and Architectures, 2005.

[15] D. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems". Proc. of 16th ACM Symposium on Parallelism in Algorithms and Architectures, 2004.

[16] J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity, and churn". Proc. of IEEE Infocom, 2005.

[17] T. Steele, V. Vishnumurthy and P. Francis. "A Parameter-Free Load Balancing Mechanism For P2P Networks". Proc. of 7th International Workshop on Peer-to-Peer Systems (IPTPS), 2008.

[18] S.Serbu, S.Bianchi, P.Kropf, P.Felber. "Dynamic Load Sharing in Peer-to-Peer Systems: When Some Peers Are More Equal than Others". IEEE Internet Computing, vol. 11, no. 4, pp. 53-61, 2007

[19] R. Jain, W. Hawe and D. Chiu. "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems". DEC-TR-301, 1984.

[20] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek and R. Morris. "Designing a DHT for low latency and high throughput". Proc. of 1st Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[21] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy and J. Zahorjan. "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload" Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP), 2003.