5.2.3 Distributed Hash Tables

Let us now take a closer look at recent developments on how to resolve an identifier to the address of the associated entity. We have already mentioned distributed hash tables a number of times, but have deferred discussion on how they actually work. In this section we correct this situation by first considering the Chord system as an easy-to-explain DHT-based system. In its simplest form, DHT-based systems do not consider network proximity at all. This negligence may easily lead to performance problems. We also discuss solutions for network-aware systems.

General Mechanism

Various DHT-based systems exist, of which a brief overview is given in Balakrishnan et al. (2003). The Chord system (Stoica et al., 2003) is representative for many of them, although there are subtle important differences that influence their complexity in maintenance and lookup protocols. As we explained briefly in Chap. 2, Chord uses an \( m \)-bit identifier space to assign randomly-chosen identifiers to nodes as well as keys to specific entities. The latter can be virtually anything: files, processes, etc. The number \( m \) of bits is usually 128 or 160, depending on which hash function is used. An entity with key \( k \) falls under the jurisdiction of the node with the smallest identifier \( id \geq k \). This node is referred to as the successor of \( k \) and denoted as \( succ(k) \).

The main issue in DHT-based systems is to efficiently resolve a key \( k \) to the address of \( succ(k) \). An obvious non-scalable approach is let each node \( p \) keep track of the successor \( succ(p+1) \) as well as its predecessor \( pred(p) \). In that case, whenever a node \( p \) receives a request to resolve key \( k \), it will simply forward the request to one of its two neighbors—whichever one is appropriate—unless \( pred(p) < k \leq p \) in which case node \( p \) should return its own address to the process that initiated the resolution of key \( k \).

Instead of this linear approach toward key lookup, each Chord node maintains a finger table of at most \( m \) entries. If \( FT_p \) denotes the finger table of node \( p \), then

\[
FT_p[i] = succ(p+2^{i-1})
\]

Put in other words, the \( i \)-th entry points to the first node succeeding \( p \) by at least \( 2^{i-1} \). Note that these references are actually short-cuts to existing nodes in the identifier space, where the short-cutted distance from node \( p \) increases exponentially as the index in the finger table increases. To look up a key \( k \), node \( p \) will then immediately forward the request to node \( q \) with index \( j \) in \( p \)'s finger table where:

\[
q = FT_p[j] \leq k < FT_p[j+1]
\]

or \( q = FT_p[1] \) when \( p < k < FT_p[1] \). (For clarity, we ignore modulo arithmetic.)