$P = 1/2^n$, a value that decreases exponentially with n (i.e. with the "running time") as required in any good randomized algorithm. We do not pretend that our proposal on how to conduct a trial is perfect, but after all we do better than Bridlegoose who gave a correct sentence every other case.

5.5 An interesting example: file sharing on the Internet

Randomized algorithms have been developed by computer scientists for solving their own problems that, although less dramatic than sentencing a defendant to death, are nevertheless worth being studied. As an example we pick the problem of file sharing in the Internet as one of the most interesting, simplifying its mathematical analysis as much as possible. The overall structure is the one of a group of users with equal resources and rights (*peers*) connected to a network where they download files of interest directly from each other. Today this type of communication is mostly devoted to getting musical files in MP3 format, and hashing is a standard technique to allocate and retrieve these files.

The most popular peer-to-peer file sharing systems are decentralized, that is they are fully distributed among the users. There is no central server hosting a complete set of addressing tables to route a message between users, rather each user stores routing information locally in limited amount.[17] Furthermore all users execute the same protocols for locating and downloading files. To do it efficiently the files must be stored at the user sites as evenly as possible, coping with the non trivial problems of relocating the files of a user who gets out of the system, or assigning files to a new user. In fact standard hash functions are very efficient for storing and retrieving items in a static environment, however, if the storage buckets change dynamically, nearly all items must be relocated at each change. The problem is then finding a scheme where most of the items remain in their buckets after the insertion or the removal of one bucket from the system. To this end a randomized allocation algorithm called *consistent hashing* works nicely, but the results are possibly incorrect with a very low probability. The algorithm was originally designed for distributing files in a dynamic family of Web servers, and is currently used in a

[17] According to the way is conducted, file sharing may raise legal problems that are not our role to investigate. The decentralized structure was originally chosen to circumvent these problems.

wealth of distributed applications to cope with system changes and failures.

Roughly speaking the problem is allocating evenly M items (files) into N buckets (peer users) connected in a network such that a change in the set of buckets requires only M/N items to be relocated on average (note that M/N is the expected number of items of one bucket). Furthermore this must be associated with an efficient method for file retrieval. Consistent hashing meets the first requirement in a much finer way.

Chosen an integer C to represent the maximum number of buckets allowed to participate into the game, the whole system can be represented on a circle that hosts the interval [0,C-1], wrapped clockwise. The buckets and the items are randomly mapped to the integers in [0,C-1], then buckets and items are represented as points on the circle. An elementary example is given in figure 5.5 with only $N = 10$ buckets and $M = 10$ items out of a maximum of 256 each. Each item is allocated into the closest bucket encountered clockwise around the circle.A new bucket entering the system is mapped randomly on the circle and receives proper items from the successor. A leaving bucket sends all its items to the successor.

Since the mapping of buckets and items to the circle is random, the expected number of items per bucket is M/N. Therefore roughly $M/2N$ items are inserted into a new bucket, and M/N items are released by a leaving bucket. Since these item relocations take place between two buckets only, while a participation of more buckets would ensure a more balanced distribution, consistent hashing makes use of a more complex strategy where buckets are replicated randomly in several copies along the circle. With this and other details that we skip here the whole algorithm attains some important results that we summarize below, taking in mind what follows.

As the number of buckets is continuously changing, N is the value relative to each *view* of the system, i.e. to the set of buckets existing at each moment. We assume $N \geq C/k$ for some constant k, i.e. each view must contain at least a certain fraction of all the possible buckets. For a given item i, the *spread* $\sigma(i)$ is the number of different buckets to which i is mapped over all the views, and the spread σ of the distribution is the maximum spread among the items. For a given bucket b, the *load* $\lambda(b)$ is the number of different items assigned to b over all the views, and the load λ of the distribution is the maximum load among the buckets. Note that spread and load

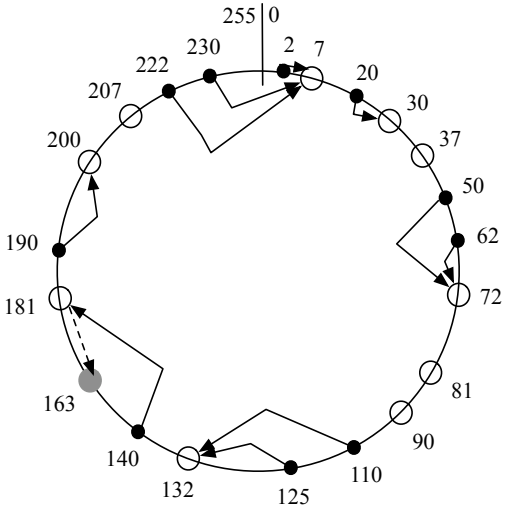

Figure 5: The circle of consistent hashing for $C = 2^8 = 256$. Buckets and items are represented with white circles and black dots, respectively. Solid arrows indicate the destination of items. The grey circle 163 indicates a new bucket receiving items from its successor 181 (dashed arrow).

are strong indicators of the quality of the distribution, and should be kept as low as possible. For consistent hashing we have:

- the protocol is *fast*, i.e. the expected time to map an item to a bucket is of order $O(1)$ (constant), and to add or delete a bucket is of order $O(\log C)$;
- the distribution is *balanced*, i.e. the probability that any given item i goes to any given bucket b is $1/N$;
- σ and λ are of order $O(k \log C)$ with probability $\geq 1 - 1/C$.

The values of σ and λ depend on the size of the views through the value of k (large views imply small k); and depend modestly on the maximum number of buckets as they grow with $\log C$. In fact C is regarded as a free parameter. The probability $1/C$ that σ or λ exceeds the expected order of magnitude decreases exponentially if the term $\log C$ increases. For example increasing C from 2^n to 2^{n+k}

the probability of error is divided by 2^k while $\log_2 C$ grows only to $n + k$.

Let us now see how consistent hashing can be used for file retrieval in a peer-to-peer system. We assume that a bucket is identified by the Internet address (or IP address) of the corresponding user and an item is identified by the name of the corresponding file.[18] A standard choice is the adoption of the hash function SHA-1 to map users and files to the circle, so up to $C = 2^{160}$ points are usable. For simplicity we take $C = 2^8$in the examples, together with a random mapping R to the interval [0,255]. So a user with IP address $A(u)$ is mapped to $R(A(u)) = 132$; and the MP3 file of "Isla Bonita is mapped to $R(IslaBonita) = 110$ (from now on users and files will be denoted by their positions). Again files are assigned to the closest user clockwise, so in figure 5.5 file 110 would be stored at user 132. If a user joins or leaves the system, files are relocated as explained before. Let us see now how lookup is done. For a user u looking for file f the general strategy is the following:

1. u sends the request to p that is the closest predecessor of f;
2. p passes the request to its successor s which contains f;
3. s sends f to u whose IP address is contained in the request.

Referring to figure 5.5, user $u = 200$ looking for file $f = 110$ must send the request to $p = 90$ (predecessor of 110), and 90 passes the request to $s = 132$ which contains the file. 132 then sends f to 200. The problem is how each user knows the positions of the other users on the circle to find predecessors and successors. In the example, how 200 finds 90, and how 90 finds 132.

Since the protocol is intended for a distributed system the users cannot interrogate a central server to know the positions of their peers and then this information must be stored in the users themselves. For $N \leq 2^m$ a good tradeoff between storage space and search speed is obtained by assigning to each user a set of m positions (*fingers*) of other users, together with their IP addresses where to send a

[18] In particular we refer to a major file retrieval service called *Chord*, described here in its main lines. For a complete description of Chord see the bibliographical notes. Note that, unlike in consistent hashing, users are now mapped only once on the circle. Recall that an IP address is a number associated with each computer directly connected to the network, see the following chapters.

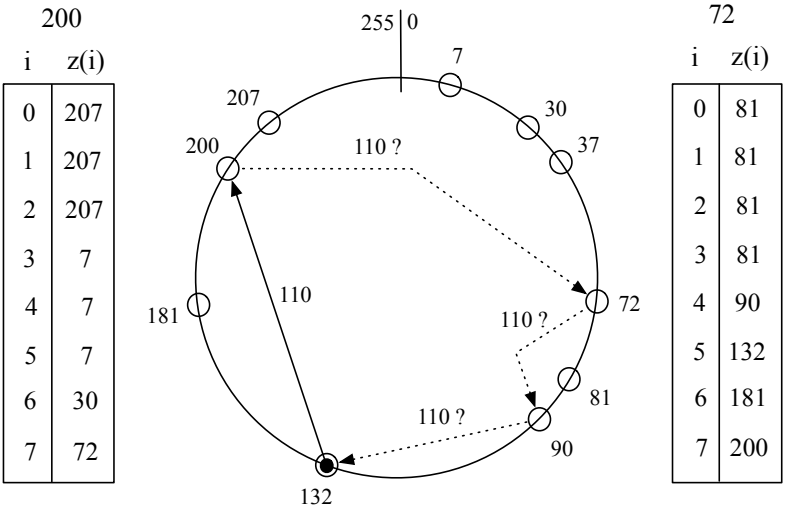

Figure 6: Fingers z(i) for users 200 and 72. User 200 retrieves Isla Bonita (file 110 stored at user 132) using these fingers.

message.[19] For $i = 0$ to m-1, the finger $z(i)$ of user u is the position of the user closest to $w(i) = u + 2^i$ clockwise, where the addition is taken modulo C. The fingers for users 72 and 200 are shown in figure 5.6. E.g. for user 72 we have:

$w(0) = (72 + 2^0) \bmod 256 = 73$, hence $z(0) = 73$
(note that $z(0)$ is always the successor of u);

$w(4) = (72 + 2^4) \bmod 256 = 86$, hence $z(4) = 90$;

$w(6) = (72 + 2^6) \bmod 256 = 138$, hence $z(6) = 181$; etc.

For user 200 we have:

$w(1) = (200 + 2^1) \bmod 256 = 202$, hence $z(1) = 207$;

$w(6) = (200 + 2^6) \bmod 256 = 8$, hence $z(6) = 30$;

$w(7) = (200 + 2^7) \bmod 256 = 72$, hence $z(7) = 72$; etc.

To look for a file f, user u sends a request to the user x that, according to its fingers, is the closest predecessor of f, and asks x

[19] Recall that N changes continuously so we can only fix an upper bound for it.

to look for f. As u has knowledge of only $m \sim \log_2 N$ of its peers, more than likely x is not the real closest predecessor of f. The request then goes on from x in the circle with the same strategy, and proceeds through other users until f is found and sent back to u. The previous example may be followed on figure 5.6. To retrieve file 110, user 200 searches for the predecessor of 110 among its fingers finding 72 to which the request is sent (recall that IP addresses are stored with the fingers). 72 in turn looks for the predecessor of 110 among its fingers and sends the request to 90. Since the successor of 90 is 132 ¿ 110 (i.e. 90 is the real closest predecessor of 110) the request is sent to 132 that has the file and sends it back to 200. Assuming that the only user 200 likes nostalgia favorites of the 1940s and looks for an MP3 version of "Mona Lisa" with hash $R(MonaLisa) = 128$, lookup would follow the same steps up to user 132 (successor of 128) that sends back a negative answer.

The reader may realize how this algorithm recalls the one of *binary search* explained in Chapter 4. Due to the balanced distribution of consistent hashing, the stretch of the circle on which f resides is at least halved at each iteration with high probability. So the request makes at most m hops in the network and the expected time for one lookup is $O(\log N)$.[20] Finally it is worth noting that a limited number of user changes do not affect the system too much, so that finger tables may be kept for long before being updated. In addition the system is difficult to disconnect with random changes because each user has $O(\log N)$ connections to other users.

5.6 Randomness and the humans (instead of computers)

We have started this chapter talking of gambling and divination as the most natural human activities having to do with randomness. Now that we know much more on random phenomena we may think some more over our perception of them.

First, in the history of man divination has been more important than one may suspect. The ancient populations of Eurasia, and later of North America, held divinatory rites to direct hunting expeditions along paths were game could be found. Such rites had a dramatic

[20]The search for a predecessor in a finger table can be done in additional $O(\log m) = O(\log \log N)$ time with binary search, that leaves the overall lookup time unchanged. Of course our presentation of the protocol has been oversimplified. In particular insertions and deletions of users require $O(\log^2 N)$ time.