Multiple Choice Tries and Distributed Hash Tables

Luc Devroye* Gábor Lugosi† Gahyun Park‡ Wojciech Szpankowski §

May 4, 2006

Abstract

Tries were introduced in 1960 by Fredkin as an efficient method for searching and sorting digital data. Recent years have seen a resurgence of interest in tries that find applications in dynamic hashing, conflict resolution algorithms, leader election algorithms, IP addresses lookup, Lempel-Ziv compression schemes, and distributed hash tables. In some of these applications, most notably in *distributed hash tables* one needs to design a well balanced trie, that is, a trie with the height as close as possible to its fillup level. In this paper we consider tries built from n strings such that each string can be chosen from a pool of k strings, each of them generated by a discrete i.i.d. source. Three cases are considered: $k = 2$, k is large but fixed, and $k \sim c \log n$. The goal in each case is to obtain tries as balanced as possible. Various parameters such as height and fill-up level are analyzed. It is shown that for two-choice tries a 50% reduction in height is achieved when compared to ordinary tries. In a greedy on-line construction when the string that minimizes the depth of insertion for every pair is inserted, the height is only reduced by 25%. In order to further reduce the height by another 25%, we design a more refined on-line algorithm. The total computation time of the algorithm is $O(n \log n)$. Furthermore, when we choose the best among $k \geq 2$ strings, then for large but fixed k the height is asymptotically equal to the typical depth in a trie, a result that cannot be improved. Finally, we show that further improvement can be achieved if the number of choices for each string is proportional to $\log n$. In this case for unbiased memoryless sources highly balanced trees can be constructed by a simple greedy algorithm for which the difference between the height and the fill-up level is bounded by a constant with high probability. This, in turn, has implications for distributed hash tables, leading to a randomized ID management algorithm in peer-to-peer networks such that, with high probability, the ratio between the maximum and the minimum load of a processor is $O(1)$.

*School of Computer Science, McGill University, 3450 University Street, Montreal H3A 2K6, Canada, luc@cs.mcgill.ca. This research was sponsored by NSERC Grant A3456 and FQRNT Grant 90-ER-0291.

†ICREA and Department of Economics, Universitat Pompeu Fabra, Ramon Trias Fargas 25-27, 08005 Barcelona, Spain, lugosi@upf.es. This research was sponsored by the Spanish Ministry of Science and Technology and FEDER, grant BMF2003-03324 and by the PASCAL Network of Excellence under EC grant no. 506778.

‡Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, Indiana, 47907-2066, USA, gpark@cs.purdue.edu.

§Department of Computer Sciences, Purdue University, 250 N. University Street, West Lafayette, Indiana, 47907-2066, USA, spa@cs.purdue.edu. This research was sponsored by NSF Grants CCR-0208709, CCF-0513636, and DMS-0503742, AFOSR Grant FA8655-04-1-3074, and NIH Grant R01 GM068959-01.

1 Introduction

A trie is a digital tree built over n strings (see Knuth (1997), Mahmoud (1992) and Szpankowski (2001) for an in-depth discussion of digital trees.) A string is stored in an external node of a trie and the path length to such a node is the shortest prefix of the string that is not a prefix of any other strings.

Tries are popular and efficient data structures that were initially developed and analyzed by Fredkin (1960) and Knuth (1973) as an efficient method for searching and sorting digital data. Recent years have seen a resurgence of interest in tries that find applications in dynamic hashing, conflict resolution algorithms, leader election algorithms, IP address lookup, Lempel-Ziv compression schemes, and others. Distributed hash tables (cf. Malkhi et al. (2002) and Adler et al. (2003)) arose recently in peer-to-peer networks in which keys are partitions across a set of processors. Tries occur naturally in the area of ID management in distributed hashing, though they were never explicitly named. One of the major problems in peer-to-peer networks is load balancing. We address this problem by redesigning old-fashioned tries into highly balanced trees that in turn produce an $O(1)$ balance (i.e., the ratio between the maximum and the minimum load) in the partition of processors in such networks. We accomplish this by adopting the "power-of-two" technique (cf. Azar, Broder, Karlin and Upfal (1994, 1999)) that already found many successful applications in hashing.

We consider random tries over $\mathcal{N}$, the set of positive integers, where each datum consists of an infinite string of i.i.d. symbols drawn from a fixed distribution on $\mathcal{N}$. The probability of the i-th symbol is denoted by p_i. The tries considered here are constructed from n independent strings $X_1, \ldots, X_n$. Each string determines a unique path from the root down in an infinite $\mathcal{N}$-ary tree: the symbols have the indices of the child nodes at different levels, that is, the path for X_i starts at the root, takes the X_{i1}-st child, then the X_{i2}-st child of that node, and so forth. Let N_u be the number of strings traversing node u in this infinite tree. A string is associated with the node u on its path that is nearest to the root and has $N_u = 1$. The standard random trie for n strings consists of these n marked nodes, one per string, and their paths to the root. The marked nodes are thus the leaves (external nodes) of the tree.

The properties of the standard random trie are well-known (see Szpankowski, 2001): for example, if D_n is the depth of a random leaf (i.e., its path distance to the root), then $D_n \sim 1/H \log n$ (in probability) as $n \to \infty$, where $H = -\sum_i p_i \log p_i$ is the entropy of the distribution. This result remains true even if $H = \infty$. The mean and variance of D_n were first obtained by Jacquet and Régnier (1986), Pittel (1985) and Szpankowski (1988).

If H_n denotes the height of the trie, i.e., the maximal distance between root and leaves, then $\frac{H_n}{\log n} \to \frac{2}{Q}$ in probability as $n \to \infty$, where $Q = -\log\left(\sum_i p_i^2\right)$ (Pittel, 1985). From Jensen's inequality and $(\max_i p_i)^2 \le \sum_i p_i^2 \le \max_i p_i$, we have

$$\frac{1}{H} \le \frac{1}{Q} \le \frac{1}{\log\left(\frac{1}{\max_{i\ge 1} p_i)}\right)} \le \frac{2}{Q},$$

so that the height is always at least twice as big as the typical depth of a node.

In some applications, it is important to reduce the height as much as possible. Attempts in this direction include PATRICIA trees (Morrison, 1968) and digital search trees (Coffman and Eve, 1970, Konheim and Newman, 1973). In PATRICIA trees, all internal nodes with one child are eliminated. In digital search trees, each internal node is associated with a string, namely the first string that visits that node (the order of $X_1, \ldots, X_n$ thus matters). In both cases, we have $\frac{H_n}{\log n} \to \frac{1}{\log\left(\frac{1}{\max_{i\ge 1} p_i)}\right)}$ in probability (Pittel, 1985), however, "in order traversal" of

digital search trees does not visit the nodes in sorted order, and internal edges of PATRICIA trees have cumbersome labels.

The height is not the only balance parameter. In a trie with finite alphabet size β (i.e., X_i takes values on $\{1, \ldots, \beta\}$), the fill-up level F_n, the distance from the root to the last level that has a full set of β^{F_n} nodes, is also important. Pittel (1986) found the typical value of F_n in a trie built over n strings generated by mixing sources. For memoryless sources, $\frac{F_n}{\log n} \to \frac{1}{\log(1/p_{\min})} = \frac{1}{h_{-\infty}}$ in probability where $p_{\min} = \min_i\{p_i\}$ is the smallest probability of generating a symbol and $h_{-\infty} = \log(1/p_{\min})$ is the Rényi entropy of infinite order (Szpankowski (2001)). This was further extended by Pittel (1986), Devroye (1992), and Knessl and Szpankowski (2005) who proved that the fill-up level F_n is concentrated on two points k_n and $k_n + 1$, where for asymmetric sources k_n is an integer $(\log n - \log\log\log n)/\log(1/p_{\min}) + O(1)$ while for symmetric sources (i.e., sources with $p_1 = p_2 = 1/2$) k_n is $\log_2 n - \log_2\log_2 n + O(1)$. Observe that in the symmetric case we have $\log\log n$ instead of $\log\log\log n$.

In many applications, most notably in *distributed hash tables* (cf. Malkhi et al. (2002) and Adler et al. (2003)), one needs to construct a well balanced trie, that is, a trie with the height as close to its fillup level as possible. As the first step we propose the so called two-choice trie in which we deal with pairs of strings. For every pair of strings we actually insert in the trie the one that has smaller depth of insertion. We first show in Theorem 1 that

$$\frac{H_n}{\log n} \to \frac{3}{2Q} \text{ in probability,}$$

resulting in a 25% reduction in height compared to standard tries. To reduce the height further, we design a refined version of the power-of-two tries in which one selects n strings resulting in a height that is close to the smallest possible. Call this height H_n^*. We design an on-line algorithm with total computational time $O(n \log n)$ such that (cf. Theorem 2)

$$\frac{H_n^*}{\log n} \to \frac{1}{Q} \text{ in probability.}$$

Interestingly, one can further reduce the height by considering tries with k choices for large k. We prove in Theorem 5 that if k is sufficiently large (but fixed), then the ratio $H_n^*/\log n$ approaches $1/H$ with arbitrary precision, a bound that cannot be improved.

As discussed above, tries occur naturally in the area of ID management in distributed hash tables. In this area of network research, n hosts are assigned one ID in the unit interval $[0, 1)$. The unit interval is wrapped around to form a ring. At any time, the set of ID's partition $[0, 1)$ into n intervals on the ring. The hosts are organized in the ring, linking to the next smaller and next larger host, and are in general quite unaware of the ID's of the other hosts, except what can be gleaned from traversing the ring in either direction. Hosts are added and deleted in some way, but ID choices are up to the system. Each interval is "owned" by the host to its left; see Figure 1. Two parameters are of some importance here. The first one is the balance B_n in the partition, as determined by the ratio of the lengths of the largest to the smallest interval. Secondly, one must be able to determine quickly which host owns an interval in which a given ID x falls. The latter is the equivalent of a search operation.

ID's are represented by their (infinite) binary expansions. Assume, for example, that the n ID's are i.i.d. and uniformly distributed on $[0, 1)$. See, e.g., Ratnasamy et al (2001), Malkhi et al (2002) or Manku et al (2003). Then it is a routine exercise in probability to show that the largest spacing defined by the ID's is asymptotic to $\log n/n$ in probability and that the smallest spacing is $\Theta(1/n^2)$ in probability (see Lévy (1939) or Pyke (1965)). Thus, B_n is asymptotic to $n \log n$ in probability. Adler et al (2003) and Naor and Wieder (2003) implicitly suggest a

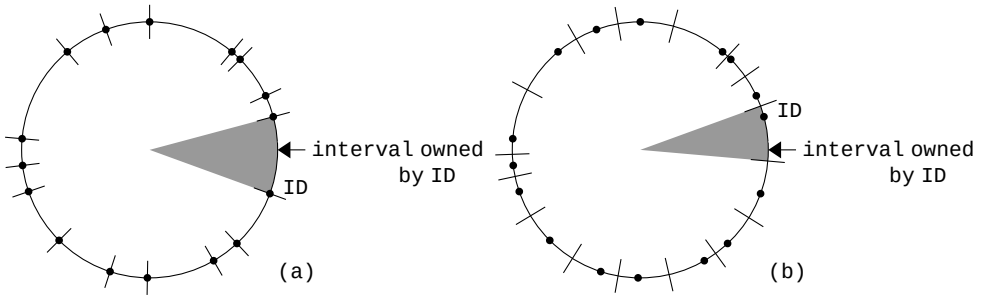

Figure 1: Two ways of partitioning the space. In both (a) and (b), IDs are randomly generated on the perimeter. In (a), IDs own the intervals to their left in clockwise order. In (b), IDs own an interval whose boundaries are determined in some manner, e.g., by virtue of a trie or digital search tree. A downloader or user generates a random number X on the perimeter and picks the owner of the interval of X for its job. The objective is to make all intervals of about equal length, so that all hosts receive about equal traffic

digital search tree approach. Consider first a trie approach (not considered in those papers): the ID's are considered as strings in a binary trie (with $p_1 = p_2 = 1/2$), and ID's are inserted sequentially as in a digital search tree. The binary expansions of the nodes that are associated with the ID's are the actual ID's used (so, each ID is mapped to another one). In the case of a trie, each leaf is associated with an ID. In the case of a digital search tree, each internal node is mapped to an ID. That means that the ID's used have only a finite number of ones in their expansions. If the height of the trie is H_n, and the fill-up level (the number of full levels in the trie) is F_n, then $B_n = 2^{H_n - F_n + O(1)}$. Since $H_n = 2\log_2 n + O(1)$ in probability and that $F_n = \log_2 n - \log_2 \log_2 n + O(1)$, hence $B_n = \Theta(n \log n)$ in probability. However, for a digital search tree, we have $H_n = \log_2 n + O(1)$ in probability, while F_n is basically as for tries. Thus, $B_n = O(\log n)$ in probability. This is essentially the result of Adler et al (2003) and Naor and Wieder (2003).

In an attempt to improve this, Dabek et al (2001) proposed attaching $b = \log n$ randomly generated ID's to each host, swamping the interval, and then making the ID assignments to guarantee that $B_n = O(1)$ in probability. However, some discard this solution as too expensive in terms of resources for maintenance.

Abraham et al (2003), Karger and Ruhl (2003) and Naor and Wieder (2003) achieve $B_n = O(1)$ in probability while restricting hosts to one ID. In another approach, Abraham et al (2003) and Naor and Wieder (2003) pick $\log n$ i.i.d. uniform random numbers per host, and assign an ID based on the largest interval these fall into: the largest interval is split in half. A little thought shows that this corresponds to a digital search tree in which $\log n$ independent strings are considered, and only one is selected for insertion, namely the one that would yield a leaf nearest to the root. The fact that both H_n and F_n are now $\log_2 n + O(1)$ in probability yields the result. Manku (2004) proposed a digital search tree with perfect balancing of all subtrees of size $\log_2 n$. It also has $B_n = O(1)$ in probability. A similar (but different) view can also be taken for the trie version: start with an ordinary binary trie with the modification (see, e.g., Pittel, 1985) that leaf nodes are mapped to their highest ancestors that have subtrees with $b = \log_2 n$ or fewer leaves. These ancestors are the leaves of the so-called ancestor trie. Construct such a binary b-trie and its ancestor trie from n i.i.d. uniform $[0, 1)$ random numbers. Now, for each ancestor, partition its interval equally by spreading the leaves in its subtree out evenly when associating ID's. We know from the Erdős-Rényi law of large numbers that the

maximal k-spacing (with $k = c \log n$) determined by n i.i.d. uniform $[0,1)$ random numbers is $\Theta(\log n/n)$ in probability (Erdős and Rényi, 1970; Deheuvels, 1985; see also Novak, 1995). This would imply that all ancestors are at level $\log_2 n - \log_2 \log_2 n + O(1)$ in probability, and that all intervals owned by the ID's are $\Theta(1/n)$ in probability, from which $B_n = O(1)$ in probability.

The power-of-two choices can be explored in the present context. If we make an ordinary trie by taking the best of two ID's as described in this paper, and then map ID's to the strings that correspond to the corresponding leaf values, then $H_n = \log_2 n + O(1)$ in probability. However, it is easy to verify that F_n is as for the standard binary trie, so that $B_n = O(\log n)$ in probability. Indeed, the largest gap defined by $2n$ uniform strings on $[0,1]$ is still $\Theta(\log n/n)$ in probability.

In this paper, using our findings from multiple-choice tries, we will prove in Theorem 7 that $H_n - F_n \leq 2$ with probability tending to one provided the number of choices per datum is proportional to $\log n$. However, we have two modifications: first, we insist on using tries instead of digital search trees; and secondly, because of the use of tries, we have to modify the selection heuristic as picking the largest interval is not good enough for tries. Furthermore, in Theorem 9 we show that by a natural *greedy on-line algorithm* one also achieves nearly perfect balancing with $H_n - F_n \leq 7$ with probability $1 - o(1)$. This has applications for load balancing in peer-to-peer networks. In particular, the result implies that if in a peer-to-peer network in which ID's of the n hosts are organized on a circle and upon arrival, each host is allowed to try $c \log n$ randomly chosen ID's and choose the one that maximizes its distance from its neighbors then the maximal load balance ratio remains bounded with high probability.

2 Two-choice tries

In this section we consider the situation when each datum has two independent strings X_i and Y_i drawn from our string distribution, and that we are free for pick one of the two for inclusion in the trie.

2.1 A simple greedy heuristic

We start with analyzing the possibly simplest algorithm when we greedily pick one string according to a simple rule: choose the string which, at the time of its insertion would yield the leaf nearest to the root. Once a selection is made, it is impossible to undo it at a later time. This greedy heuristic yields a height that is guaranteed to be 25% better than that of the ordinary trie, but it cannot achieve the 50% improvement of the main method described below. Here, H_n refers to the height of the trie obtained by this greedy heuristic. We can prove the following result.

Theorem 1 *For all integer $d > 0$,*

$$\mathbb{P}\{H_n \geq d\} \leq 4n^3 e^{-2dQ} + 2n^2 e^{-3dQ/2}.$$

Thus, for any $t > 0$,

$$\mathbb{P}\left\{H_n \geq \frac{3\log n + t}{2Q}\right\} \leq 4e^{-t} + 2n^{-1/4} e^{-3t/4}.$$

On the other hand, if that there exists an integer β such that $p_1 = \cdots = p_\beta = 1/\beta$, then, for all $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}\left\{H_n \leq \frac{(3-\epsilon)\log n}{2Q}\right\} = 0.$$

2.2 A refined algorithm

In this section we show that it is possible to gain a further significant improvement in the height by an algorithm that is more complex than the greedy heuristic discussed above. We start by showing that the best possible height is about $\log n/Q$ and then we construct an on-line algorithm of expected complexity $O(n \log n)$ which achieves this optimal order. Define $Z_i(0) = X_i, Z_i(1) = Y_i$, let $\{i_1, \ldots, i_n\} \subseteq \{0,1\}^n$, and let $H_n(i_1, \ldots, i_n)$ denote the height of the trie for $Z_1(i_1), \ldots, Z_n(i_n)$. Thus, with n data pairs, we have 2^n possible random tries. Let $H_n^* = \min_{i_1,\ldots,i_n} H_n(i_1, \ldots, i_n)$ be the minimal height over all these 2^n tries.

We show the following:

Theorem 2 *Assume that the vector of p_i's is nontrivial ($\max_i p_i < 1$). Then $H_n^*/\log n \to 1/Q$ in probability. In particular, for fixed $t \in \mathbb{R}$,*

$$\mathbb{P}\left\{H_n^* \geq \frac{\log n \ + t}{Q}\right\} \leq 8e^{-t}.$$

Also, for all $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}\left\{H_n^* \leq \frac{(1-\epsilon)\log n}{Q}\right\} = 0.$$

This theorem shows that the asymptotic improvement over standard random tries is 50%. Furthermore, the height is less than or equal to the height of the corresponding PATRICIA and digital search trees. Below we sketch the proof of the upper bound. The proof of the lower bound is omitted, and can be found in the journal version of this paper.

2.3 The upper bound

In the infinite trie formed by all $2n$ strings, we consider all subtrees T_j, $j \geq 1$ rooted at nodes at distance d from the root. We sometimes write $T_j(d)$ to make the dependence upon d explicit. More often, we just use T_j. We say that a string visits T_j if the root of T_j is on the path of the string. Prune this forest of trees by keeping only those that contain at least two leaves. Define $\lambda = \sum_i p_i^2$. The following lemma is immediate from the definition of the trie.

Lemma 3 (i) *A bad datum is one in which both of its strings fall in the same T_j. The probability that there exists a bad datum anywhere is not more than $n\lambda^d$.* (ii) *A colliding pair of data is such that for some $j \neq k$, each datum in the pair delivers one string to T_j and one string to T_k. The probability that there is a colliding pair of data anywhere is not more than $2n^2\lambda^{2d}$.*

Next, we consider a multigraph $G(d)$ (or just G), whose vertices represent the $T_j(d)$'s. We connect T_j with T_ℓ if a datum deposits one string in each of these trees (cf. Figure 2). With T_j we keep a list of indices of data for which at least one of the two strings visits T_j.

Lemma 4 *The probability that G has a cycle of length ≥ 3 is not more than $\frac{(4n)^3\lambda^{3d}}{1-4n\lambda^d}$.*

Finally, by finding thhe smallest d such that there is no bad datum, no colliding data and no cycle in $G(d)$ (so that G is a forest with no multiedges), we can assign strings as follows. For each tree in turn pick any node as the root. Then choose any one of the strings in the root node's list. For all other strings in the root's list, choose the companion string of the same datum (by following edges away from the root). This either terminates, or has an impact on one or more child trees. But for the child tree of the root, we have fixed one string (as we did

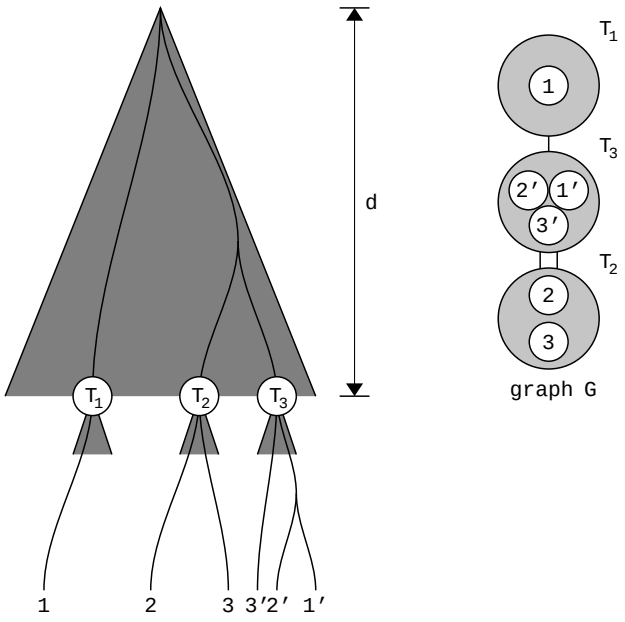

Figure 2: The multigraph G and an infinite trie for $n = 3$ pairs of strings, denoted by $(1, 1')$, $(2, 2')$ and $(3, 3')$. Note that $(2, 2')$ and $(3, 3')$ is a colliding pair.

for the root), and thus choose again companion strings for that child list, and so forth. This process is continued until one string of each datum is chosen for the trie. In this manner, the height H_n of the trie for the data selected by this procedure is not more than d. Therefore,

$$\begin{aligned}\mathbb{P}\{H_n > d\} \quad &\leq \quad \mathbb{P}\{\text{there exists a bad datum}\} + \mathbb{P}\{\text{there exists a colliding pair}\} \\ &\quad +\mathbb{P}\{\text{there exists a cycle}\} \leq n\lambda^d + 2n^2\lambda^{2d} + \frac{(4n)^3\lambda^{3d}}{1-4n\lambda^d}.\end{aligned}$$

If we set $A = n\lambda^d$, then

$$\mathbb{P}\{H_n > d\} \leq \min\left(A + 2A^2 + \frac{64A^3}{1-4A}, 1\right) \leq 4A\mathbb{I}_{[A\leq 1/8]} + \mathbb{I}_{[A>1/8]} \leq 4A\mathbb{I}_{[A\leq 1/8]} + 8A\mathbb{I}_{[A>1/8]} \leq 8A.$$

Thus, $\mathbb{P}\{H_n > d\} \leq 8n\lambda^d$, and the upper bound of Theorem 2 is proved.

2.4 Algorithmic considerations

Both the off-line and on-line constructions of the two-choice trie achieving the upper bound in Theorem 2 can be carried out by maintaining a data structure that guarantees that the height H_n is at all times the smallest integer d such that $G(d)$ is acyclic, where multiple edges between nodes are counted as cycles. The n data pairs are stored in an array, and the infinite trie for $2n$ strings, truncated at height H_n, is stored as well. Each leaf in this trie t_n (truncated at H_n) represents in fact a subtree T_i, which in turn contains the nodes of $G = G(d)$ with $d = H_n$. To find things easily, each node of G has a pointer to a linked list of data strings. And vice versa, each string in the array of n pairs of strings has a pointer to the subtree T_i in the trie to which it belongs.

Nodes of G are organized in turn in a parent pointer data structure commonly used for managing forests (see, e.g., Tarjan, 1983), and one linked list per tree in the forest (G). The only operations needed on this structure are findroot (self-explanatory) and join (join two

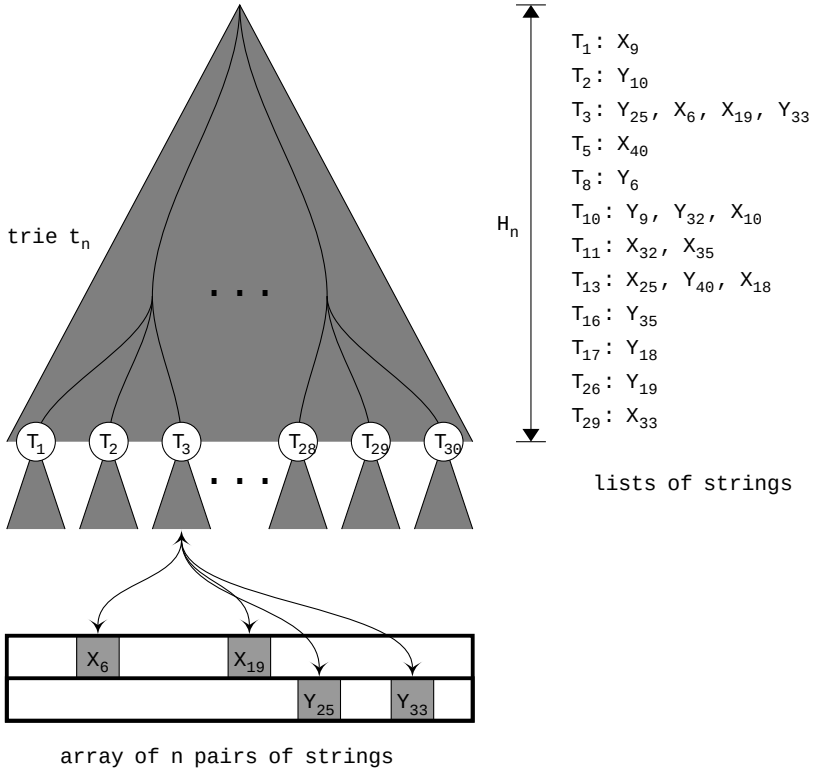

Figure 3: The basic data structure needed to efficiently construct the two-choice trie.

components). Findroot is implemented by following links to the root. We ensure that the height of the parentpointer trees is always bounded by $\log_2 n$, where n is the number of nodes in the tree. A join proceeds by making the root of the smallest of the two subtrees the child of the root of the other tree. The two linked lists of the nodes in the trees are joined as well. This takes constant time. By picking the smaller tree, we see that the height of the parent pointer tree is never more than $\log_2 n$.

Assume that we have maintained this structure with n data pairs and that the height of t_n is $h = H_n$. Then, inserting data pair $n+1$, say (X, Y), into the structure proceeds as follows: for X and Y in turn, determine the nodes of G in which they live, by following paths down from the root in t_n. Let these nodes of G be T_j and T_k. Run findroot on both nodes, to determine if they live in the same component. If they do not, then join the components of T_j and T_k, add X to the linked list of T_j, and add Y to the linked list of T_k. The work done thus far is $O(h + \log n)$.

If T_j and T_k are in the same component, then adding an edge between them would create a cycle in G. Thus, we destroy t_n and create t'_n of height $h+1$ from scratch in time $O(n)$ (see below how). An attempt is made to insert (X, Y) in t'_n. We repeat these attempts, always increasing h by one, until we are successful. The time spent here is $O(n\Delta h)$, where Δh is the number of attempts.

In a global manner, starting from an empty tree, we see that to create this structure of size n takes time bounded by $O(nH_n + n \log n)$. By Theorem 2, $\mathbb{E}\{H_n\} = O(\log n)$. Therefore, the expected time is $O(n \log n)$, which cannot be improved upon.

The space used up is $O(nH_n)$. It is known that for standard tries the expected number of internal nodes is $O(n/H)$, where H is the entropy (Régnier and Jacquet, 1989). While it

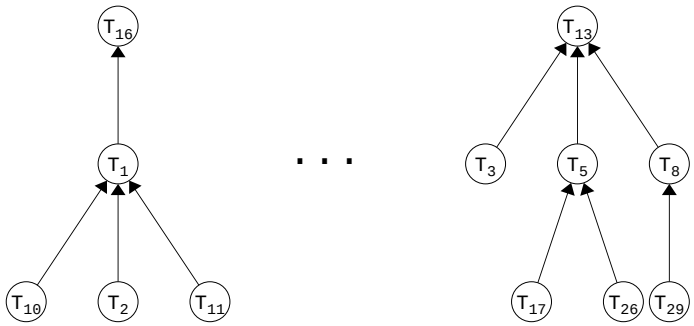

Figure 4: The forest G is maintained by organizing each tree component in a parentpointer tree. The components in the figure might correspond to the list of strings given in the previous figure

is still true that the expected number of internal nodes in the final trie is $O(n)$ (because it is smaller than that for the trie constructed using all $2n$ data strings), the intermediate structure needed during the construction is possibly of expected size of the order of $n \log n$.

Two details remain to be decided. First, we have to choose one element in each data pair. This can be done quite simply by considering the roots for all components in turn. From the root tree in a component, say T_r, pick any of its member strings, and assign it. Then traverse the component of T_r by depth first search, where the edges are the edges of G (an edge of G is easily determined from the list of strings in T_r, as each string points back to the tree T_i to which it belongs). At each new node visited, if possible, pick the first string whose sibling has not been assigned yet. This process cannot get stuck as there is no cycle in G, and it takes time $O(n)$. In Figures 3 and 4, the component whose root tree is T_{13} is traversed in this order: $T_{13}, T_3, T_8, T_{26}, T_{29}, T_5$ and T_{17}. The string assignments for the six string pairs in that 7-node component are, in order of assignment, $X_{25}, X_6, Y_6, Y_{19}, X_{33}, X_{40}$ and Y_{18}. After the assignment of all strings, it is a trivial matter to construct the final trie in time $O(nH_n)$.

The second detail concerns the extension of G and the necessary data structures when the height h is increased by one. Here we first update the trie by splitting all the trees T_j appropriately. Create the connected components by depth first search following the edges of G. This takes time $O(n)$. Set up the parent pointer data structure for each component of G by picking a root arbitrarily and making all other nodes children of the root.

The discussion above ensures that we can construct the two-choice trie incrementally in $O(n \log n)$ expected time. Also, in a dynamic setting, if the data structure defined above is maintained, then an insertion can be performed in $O(\log n)$ expected amortized time, under the assumptions of the theorems in this paper.

2.5 Multiple choice tries

If we have k choices per datum, then the height can be further reduced. However, in any case, we cannot go beyond the entropy bound, as we will prove in this section. Recall that for an ordinary trie, $D_n = o(\log n)$ in probability when $H = \infty$. We will not deal with those cases here. Let $k \geq 2$ be a fixed integer. Consider n data, each composed of k independent strings of i.i.d. symbols drawn from any distribution on $\mathcal{N}$. Let $H_n^*(k)$ denote the minimal height of any trie of n strings that takes one string of each datum. We have the following result (in fact, the lower bound holds for all $k \geq 2$).

Theorem 5 *Assume $H < \infty$. For all $\epsilon > 0$, there exists k large enough such that*

$$\lim_{n\to\infty} \mathbb{P}\left\{ \frac{(1-\epsilon)\log n}{H} \leq H_n^*(k) \leq \frac{(1+\epsilon)\log n}{H} \right\} = 1.$$

3 Distributed hash tables and ID management

Consider the interval $[0,1]$ and let $X_1, \ldots, X_n$ be n independent vectors of $k = \lceil c \log n \rceil$ i.i.d. uniform $[0,1]$ random variables $X_{i,j}$, $1 \leq i \leq n, 1 \leq j \leq k$, where $c > 0$ is a constant. We first show that we can pick $X_{1,i_1}, \ldots, X_{n,i_n}$ such that the n spacings defined by these random variables on the circular interval $[0,1]$ (the interval wrapped to a unit perimeter circle) are close to $1/n$ with high probability. Denote by M_n the maximal spacing defined by a given selection $X_{1,Z_1}, \ldots, X_{n,Z_n}$ where $Z_1, \ldots, Z_n$ are random indices defined in some manner. Let m_n denote the minimal spacing.

Lemma 6 *Let $\alpha \in (0,1)$ be fixed. Let $c \geq 2/\alpha$. Then there exists a selection $Z_1, \ldots, Z_n$ such that $X_{1,Z_1}, \ldots, X_{n,Z_n}$ satisfies, for $n \geq 8$,*

$$\mathbb{P}\left\{ \frac{1-\alpha}{n} < m_n \leq M_n < \frac{1+\alpha}{n} \right\} \geq 1 - \frac{3}{n}.$$

Theorem 7 *Let $\alpha \in (0, 1/3)$ be fixed. Let $c = 2/\alpha$. Then there exists a selection $Z_1, \ldots, Z_n$ such that the height H_n and fillup level F_n of the associated trie for $X_{1,Z_1}, \ldots, X_{n,Z_n}$ satisfy, for $n \geq 8$,*

$$\mathbb{P}\{H_n - F_n \leq 2\} \geq 1 - \frac{3}{n}.$$

PROOF. Consider the binary trie formed by the selection $X_{1,Z_1}, \ldots, X_{n,Z_n}$ of Lemma 6. If a potential node at distance d from the root is not realized, then there is a leaf at distance less than d from the root. If that leaf is at distance $d-1$, then only one string in the selection falls in the corresponding interval, which has width $1/2^{d-1}$. Thus, the maximal spacing in the selection is at least half that, or $1/2^d$. Therefore, $1/2^d \leq M_n$. Let F_n be the fill-up level, the distance to the last full level of nodes. We have $F_n = d-1$ if d is the first level with a missing node. Therefore $F_n \geq \log_2(1/M_n) - 1$. On the other hand, $H_n = h$, then at distance $h-1$, two strings in the selection visit the same node, and thus, two strings are at distance less than $1/2h - 1$ from each other. Thus, $m_n \leq 1/2^{h-1}$, or $H_n \leq \log_2(1/m_n) + 1$. If the selection is such that $(1-\alpha)/n < m_n \leq M_n < (1+\alpha)/n$, then we have

$$\lceil \log_2 n - \log_2(1+\alpha) \rceil - 1 \leq F_n \leq H_n \leq \lfloor \log_2 n - \log_2(1-\alpha) \rfloor + 1.$$

We conclude $H_n - F_n \leq 2 + \left\lfloor \log_2 \left(\frac{1+\alpha}{1-\alpha} \right) \right\rfloor$. If $\alpha < 1/3$, and $c = 2/\alpha$, then the upper bound is 2. □

Theorem 7 is an existence theorem, and is appealing since we did not even have to move or transform any of the IDs. Recall that most ID management algorithms (e.g., Manku (2004)) allow hosts to shift their ID's to obtain a better partition. The actual construction may be cumbersome, though. We have not established an algorithm, even for $b = 2$, that can achieve such a balance on-line while not sending many messages in the network.

So, let us find an on-line algorithm that achieves the super-balancing predicted by Theorem 7. Only, the spacings referred to in the definition of B_n now refer to the ID's mapped to the leftmost parts of the intervals in the trie (see below). For each of n hosts, $k = \lceil c \log n \rceil$ i.i.d.

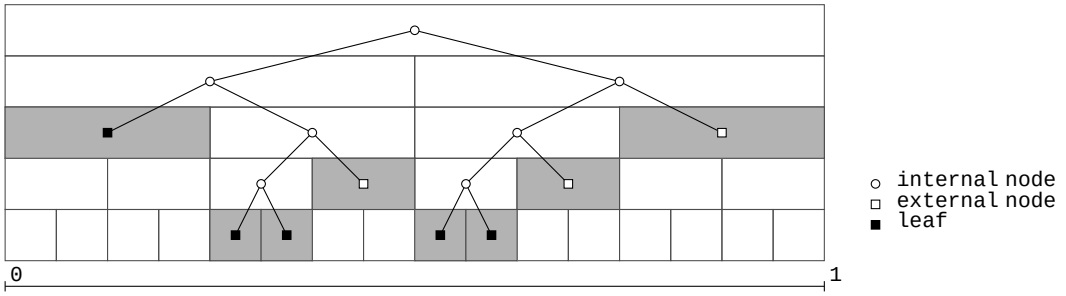

Figure 5: A standard trie for five strings. The correspondence between nodes and intervals in a dyadic partition of the unit interval is shown. The leaf ID assigned is read from the path to the root (0 for left, 1 for right). The external nodes, not normally part of the trie, are shown as well. Together, external nodes and leaves define a partition of the unit interval (shaded boxes). The fill-up level of this tree is one, while the height is four

uniform $[0,1]$ potential ID's are generated, $U_i(1),\ldots,U_i(k)$, $1 \le i \le n$. We build an ordinary trie T_n for the binary expansions of $U_1(Z_1),\ldots,U_n(Z_n)$, where the Z_i's are the selections. To define Z_{n+1}, consider the tries $T_{n,j}$ for T_n with $U_{n+1}(j)$ added, $1 \le j \le k$. This is easily done by trying to insert each ID separately. If $D_{n+1}(j)$ is the depth of the leaf of $U_{n+1}(j)$ after insertion into T_n, then $Z_{n+1} = \arg\min D_{n+1}(j)$, where ties are broken randomly. In other words, we pick the ID that at the moment of its birth has the shortest distance to the root of the trie. The actual ID assigned is implicit in the path to the root: it has a common prefix with (the binary expansion of) $U_{n+1}(j)$. This scheme generalizes the best of two-strings greedy algorithm studied above. Unlike with two choices, with $k = c\log n$ choices one does not lose optimality by a greedy construction. We first consider the height H_n and then the fillup F_n.

Theorem 8 *Let $c > 1/\log 2$ and $k = \lceil c\log n\rceil$ in the greedy heuristic for assigning IDs. Then*

$$\mathbb{P}\{H_n \ge \log_2 n + 3\} \le n^{1-c\log 2}.$$

PROOF. Given that $H_{n-1} < h$, we have $H_n \ge h$ if and only if $U_n(Z_n)$ lands within $1/2^{h-1}$ of one of $U_1(Z_1),\ldots,U_{n-1}(Z_{n-1})$. The probability of this is conservatively bounded by $\left(\frac{2(n-1)}{2^{h-1}}\right)^k$, for if one $U_n(j)$ is further than $1/2^{h-1}$ away from each $U_1(Z_1),\ldots,U_{n-1}(Z_{n-1})$, it will result in a leaf that is less than distance h away from the root. Thus, with $h = \lceil \log_2 n + 3\rceil$,

$$\mathbb{P}\left\{H_n \ge h\right\} \le \sum_{i=1}^{n} \mathbb{P}\left\{H_i \ge h | H_{i-1} < h\right\} \le n\left(\frac{1}{2}\right)^k \le n^{1-c\log 2}. \ \square$$

Theorem 9 *Let $c > (8/5)\log 2$ and $k = \lceil c\log n\rceil$ in the greedy heuristic for assigning IDs. Then*

$$\mathbb{P}\{F_n \le \log_2 n - 4\} = O(1/\log^2 n).$$

In conclusion, $H_n - F_n \le 7$ with probability tending to one when $c > 1/\log 2$. This implies that $B_n = O(1)$ in probability when spacings are defined with respect to leftmost points of leaf intervals. Finally, the trie scheme proposed here assumes that one knows n, while in distributed networks, n is unknown. However, one can estimate n by 2^D, where D is the depth of insertion of a random string in the trie: since all depths are within $O(1)$ of $\log_2 n$, the estimate is off by a constant factor only. One can thus extend the method in this manner by replacing $k = c\log n$ throughout by cD where D is the depth of insertion of a random string.

4 References

I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov, "A generic scheme for building overlay networks in adversarial scenarios," in: *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.

M. Adler, E. Halperin, R. M. Karp, and V. V. Vazirani, "A stochastic process on the hypercube with applications to peer-to-peer networks. ," in: *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003)*, pp. 575–584, 2003.

Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations (extended abstract)," in: *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pp. 593–602, 1994.

Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM Journal on Computing*, vol. 29, pp. 180–200, 1999.

H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in P2P systems," *Communications of the ACM*, vol. 1946, pp. 43–48, 2003.

H. Balakrishnan, S. Shenker, and M. Walfish, "Peering peer-to-peer providers," in: *Peer-to-Peer Systems IV, 4th International Workshop, IPTPS 2005, Ithaca, NY*, (edited by M. Castro, R. van Renesse (eds)), vol. 3640, pp. 104–114, Lecture Notes in Computer Science, Springer-Verlag, 2005.

J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, Amsterdam, 1976.

E. G. Coffman and J. Eve, "File structures using hashing functions," *Communications of the ACM*, vol. 13, pp. 427–436, 1970.

A. Czumaj and V. Stemann, "Randomized Allocation Processes," in: *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS'97), October 19-22, 1997, Miami Beach, FL*, pp. 194–203, 1997.

F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in: *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, pp. 202–215, 2001.

P. Deheuvels, "On the Erdös-Rényi theorem for random fields and sequences and its relationships with the theory of runs and spacings," *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 70, pp. 91–115, 1985.

L. Devroye, "Laws of large numbers and tail inequalities for random tries and Patricia trees," *Journal of Computational and Applied Mathematics*, vol. 142, pp. 27–37, 2002.

P. Erdös and A. Rényi, "On a new law of large numbers," *J. Anal. Math.*, vol. 22, pp. 103–111, 1970.

E. H. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, pp. 490–500, 1960.

P. Hall, "On representatives of subsets," *Journal of the London Mathematical Society*, vol. 10, pp. 26–30, 1935.

P. Jacquet and M. Régnier, "Trie partitioning process: limiting distributions," in: *CAAP 86*, (edited by P. Franchi-Zannettacci), vol. 214, pp. 196–210, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1986.

D. R. Karger and M. Ruhl, "New algorithms for load balancing in peer-to-peer systems," IRIS Student Workshop, 2003.

D. E. Knuth, *The Art of Computer Programming, Vol. 3 : Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

A. G. Konheim and D. J. Newman, "A note on growing binary trees," *Discrete Mathematics*, vol. 4, pp. 57–63, 1973.

P. Levy, "Sur la division d'un segment par des points choisis au hasard," *Comptes Rendus Acad. Sci . Paris*, vol. 208, pp. 147–149,

D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in: *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pp. 183–192, 2002.

G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," in: *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, pp. 127–140, 2003.

G. S. Manku, "Balanced binary trees for ID management and load balance in distributed hash tables," in: *23rd ACM Symposium on Principles of Distributed Computing (PODC 2004)*, pp. 197–205, 2004.

D. R. Morrison, "PATRICIA — Practical Algorithm To Retrieve Information Coded in Alphanumeric," *Journal of the ACM*, vol. 15, pp. 514–534, 1968.

M. Naor and U. Wieder, "Novel architectures for P2P applications: The continuous-discrete approach," in: *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2003)*, pp. 50–59, 2003.

S. Yu. Novak, "On the Erdős-Rényi maximum of partial sums," *Theory of Probability and its Applications*, vol. 42, pp. 254–270, 1995.

R. Pagh and F. F. Rodler, "Cuckoo hashing," BRICS Report Series RS-01-32, Department of Computer Science, University of Aarhus, 2001.

B. Pittel, "Asymptotical growth of a class of random trees," *Annals of Probability*, vol. 13, pp. 414–427, 1985.

B. Pittel, "Path in a random digital tree: limiting distributions," *Advances in Applied Probability*, vol. 18, pp. 139–155, 1986.

R. Pyke, "Spacings," *Journal of the Royal Statistical Society*, vol. 27, pp. 395–436, 1965.

S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker, "Prefix hash tree: An indexing data structure over distributed hash tables," IRB Technical Report , 2004.

S. Ratnasamy, P. Francis, M. Handley, and R. M. Karp, "A scalable content-addressable network," in: *Proceedings of the ACM SIGCOMM 2001*, pp. 161–172, 2001.

M. Régnier and P. Jacquet, "New results on the size of tries," *IEEE Transactions on Information Theory*, vol. IT-35, pp. 203–205, 1989.

A. Rényi, "On the dimension and entropy of probability distributions," *Acta Mathematica Academiae Sci. Hungarica*, vol. 10, pp. 193–215, 1959.

W. Szpankowski, "Some results on V-ary asymmetric tries," *Journal of Algorithms*, vol. 9, pp. 224–244, 1988.

W. Szpankowski, "A characterization of digital search trees from the successful search viewpoint," *Theoretical Computer Science*, vol. 85, pp. 117–134, 1991.

W. Szpankowski, "On the height of digital trees and related problems," *Algorithmica*, vol. 6, pp. 256–277, 1991.

W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*, Springer-Verlag, New York, 2001.

R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.