The Power of One Move: Hashing Schemes for Hardware

Adam Kirsch and Michael Mitzenmacher
Harvard School of Engineering and Applied Sciences
Cambridge, MA 02138
Email: {kirsch, michaelm}@eecs.harvard.edu

***Abstract*—In a standard multiple choice hashing scheme, each item is stored in one of $d \geq 2$ hash table buckets. The availability of choice in where items are stored improves space utilization. These schemes are often very amenable to a hardware implementation, such as in a router. Recently, however, researchers have discovered powerful variants where items already in the hash table may be moved during the insertion of a new item. Unfortunately, these schemes occasionally require a large number of items to be moved during an insertion operation, making them inappropriate for a hardware implementation. We show that it is possible to significantly increase the space utilization of a multiple choice hashing scheme by allowing at most one item to be moved during an insertion. Furthermore, our schemes can be effectively analyzed, optimized, and compared using numerical methods based on fluid limit arguments, without resorting to much slower simulations.**

I. Introduction

In a multiple choice hashing scheme, a hash table is built using the following approach: each item x is associated with d hash values, each corresponding to a bucket in the table, and the item is placed in one of these d locations. Such schemes are often used to ensure a more even distribution of items among the buckets of the hash table than would result from using a single hash function, and there are many theoretical results along these lines [1], [3], [12]–[14], [19]. These schemes can also be used to ensure that each bucket contains at most one item with high probability [2]. Since multiple choice hashing schemes are often very simple to implement, they have often been proposed for specific applications, such as network routers [3], peer-to-peer applications [4], and standard load balancing of jobs across machines [12], [13].

In this paper, we consider implementing multiple choice hashing schemes in hardware. From this perspective, a key property of these schemes is that during a lookup operation, if the queried item is in the hash table, then it must be in one of its d possible locations. Furthermore, in most constructions these locations can be accessed in parallel. Hence lookups are trivial, requiring one parallel (or at most a constant number of sequential) memory operations. For these reasons, multiple choice hashing schemes have been suggested for many hardware applications, such as in routers for IP lookups (e.g. [3], [18], [20]), network measurement and monitoring (e.g. [6], [18]), and other related tasks.

We seek to improve the space utilization of multiple choice hashing schemes. In particular, we consider hashing schemes that allow items already in the hash table to be moved during an insertion operation. Recently, many such schemes have been proposed in the theory literature, and the results are extremely compelling [5], [7], [15], [16]. Thus, it is natural to ask whether these proposed schemes are applicable to a hardware setting.

In general, the immediate answer is no. For example, we consider *cuckoo hashing* [15], which is the simplest multiple choice scheme that allows moves. Using cuckoo hashing, for an initially empty hash table designed to hold n items in $O(n)$ space, there is a non-negligible probability that during the insertion of n items into the table, at least one of those insertions requires $\Omega(\log n)$ moves. This is true despite the fact that, on average, only a constant number of items are moved during an insertion [5], [7], [15]. Unfortunately, in a hardware setting, the worst case time of an insertion operation may be significantly more important than the average time, because the former may essentially determine the complexity of the implementation. In particular, moving items in the hash table, which is generally held off-chip in slow memory, is expensive and must be limited. Only a small constant number of moves – indeed, arguably only one – is acceptable in practice.

A further issue with cuckoo hashing techniques is that the analysis of variants that perform well is currently incomplete. While one can construct proofs that they yield hash tables that hold n items with $O(n)$ space (with high probability), the analysis often loses significant constant factors [5], [7], [15], [16]. We emphasize that this is not just a theoretical concern, as it makes the design and optimization of such structures essentially dependent upon potentially expensive simulations. The availability of accurate theoretical results, especially in the design and optimization phase, can significantly aid in the practical development of an actual implementation.

The primary purpose of this paper is to bridge the gap between the existing theory and practice of multiple choice hashing schemes, particularly those that allow moves. We demonstrate simple but very effective multiple choice hashing schemes suitable for implementation in hardware. Specifically:

- We design and analyze schemes that require moving at most one item during each insertion operation. The limitation of just one move suggests that the schemes will be effective in practice. Moreover, we demonstrate that the gains in space utilization are substantial.
- We explicitly consider the availability of a small content-

addressable memory (CAM), and show how to optimize our schemes accordingly. While CAMs are common in hardware design to cope with rare but problematic cases, they are generally not considered in theoretical analyses.

- We use fluid limit arguments to develop numerical analysis and optimization techniques that give extremely accurate results. Our analysis allows us to consider questions such as the appropriate size for a CAM and the potential benefit of using skewed sub-tables of varying sizes in our construction. Furthermore, our approach provides a solid framework for exploring future hashing schemes.

II. The Standard Multilevel Hash Table (MHT)

A standard multilevel hash table (MHT) [2] for representing a set of n items consists of d sub-tables, $T_1, \ldots, T_d$, where T_i has $\alpha_i n$ buckets. Unless otherwise specified, we assume that a bucket can hold at most one item. For simplicity, we assume that $T_1, \ldots, T_d$ place items using independent fully random hash functions $h_1, \ldots, h_d$. To place an item x in the MHT, we simply find the smallest i for which $T_i[h_i(x)]$ is empty, and place x at $T_i[h_i(x)]$. If there is no such i, then we place x onto an *overflow list* L. In general, we visualize $T_1, \ldots, T_d, L$ as being laid out from left to right, and we often use this orientation in our discussion. (We note that in this setting, where each bucket holds at most one item, the standard MHT insertion scheme corresponds exactly to the more well-known d-left hashing scheme [3], [14], [19].) Finally, here and in most of the paper, we concern ourselves only with inserting items into an MHT, although all of what we do can be adapted to deal with deletions. We discuss this further in Section VIII.

The standard MHT is very amenable to a hardware implementation. We can easily perform a lookup by doing one hash and read for each sub-table in parallel, or sequentially if need be. In certain settings, it may even be practical to use only a single read operation, with the aid of an auxiliary summary data structure that tells us which table to read from [9]. Insertions are similarly easy to perform, as we can store a bit table in faster memory that tells us which buckets are occupied. Thus, the only operation on the actual table required during an insertion is just the writing of the item to the proper place.

The prior work on MHTs (notably [2] and [9]) considers implementations where one must be able to insert n items into some MHT with $O(n)$ buckets and be assured that, with very high probability (typically inversely polynomial in n), none will overflow into L; in effect, there is no overflow list here. Indeed, [2] shows that this can be done with an expected constant number of re-hashings of the items for $d = \log\log n + O(1)$, and [9] modifies that analysis for the case where no re-hashings are allowed, which is a much more compelling scenario for hardware applications.

We consider a different approach, designed specifically for addressing the issues that arise in hardware applications. (A similar approach, dubbed Filter Hashing, is suggested in [7].) If L is reasonably small (say, at most 64) with overwhelming probability, then we can implement it using a CAM of modest size. In this setting, it becomes useful to consider the following asymptotic regime for theoretical analysis. Rather than setting $d = \log\log n + O(1)$, we fix d to be some small constant. For example, in our analyses, we tend to focus on $d = 4$ because it gives an excellent tradeoff between performance and practicability for the reasonable value $n = 10000$. Now we think of $\alpha_1, \ldots, \alpha_d$ as being fixed constants, so that the total size of the MHT is cn buckets, for $c = \sum_{i=1}^{d} \alpha_d$. We can then imagine inserting n items into the MHT and measuring the fraction that overflow into L. It turns out that, as $n \to \infty$, this fraction is very sharply concentrated around a constant w that can be calculated from a system of differential equations. In practice, this means that for n items, if w is on the order of, say, $10/n$, then we can implement the MHT using a modest sized CAM to represent L. Furthermore, and much more importantly, this whole approach is so general that we can extend it do deal with many variations on the standard procedure for inserting items. In particular, we use this technique to show that it is possible and practical to make substantial performance improvements to the standard MHT insertion procedure simply by allowing a single item to be moved. This is the main contribution of this work.

III. Hashing Schemes and Differential Equations

It is well known that, in many situations, the behavior of a randomized hashing scheme can be effectively approximated by a deterministic system [12]. Such an approximation eliminates much of the complexity that arises in many probabilistic analyses, and also makes the results significantly more transparent. We use this approach throughout this work.

To illustrate the technique, we consider the standard MHT as described in Section II. Suppose that we start at time 0 with an empty MHT, and then insert n items into it, inserting the jth item at time j/n, so that all items are inserted by time 1. Let $F_i(t)$ denote the fraction of buckets in T_i that are occupied at time t. Now condition on the state of the MHT just after the jth item is inserted, for some $j < n$. Then, for each i, the probability that the $(j+1)$st item inserted into the MHT ends up in T_i is $(1 - F_i(j/n)) \prod_{k=1}^{i-1} F_k(j/n)$. Formally, letting $\vec{F}(t)$ denote the vector of the $F_i(t)$'s, we have

$$\begin{aligned} &\mathbf{E}[F_i((j+1)/n) - F_i(j/n) \mid \vec{F}(j/n)] \\ &= \frac{(1 - F_i(j/n))}{\alpha_i} \prod_{k=1}^{i-1} F_k(j/n). \end{aligned}$$

The conditional expectation corresponds to the average change in $F(t)$ over the window of time $[j/n, (j+1)/n]$. Thus, if n is very large, so that changes are comparatively small, then we would expect the following approximation to be valid:

$$\frac{\mathrm{d}F_i(t)}{\mathrm{d}t} \approx \frac{(1 - F_i(t))}{\alpha_i} \prod_{k=1}^{i-1} F_k(t). \qquad (1)$$

Indeed, this is essentially the case. More formally, let $\vec{f}(t) = (f_1(t), \ldots, f_d(t))$ be the solution to the system of differential equations

$$\frac{\mathrm{d}f_i}{\mathrm{d}t} = \frac{(1 - f_i)}{\alpha_i} \prod_{k=1}^{i-1} f_k$$

with $f_i(0) = F_i(0) = 0$ for each i. Then as $n \to \infty$ and $\epsilon \to 0$ we have,

$$\mathbf{Pr}\left(\sup_{t\in[0,1]}\left|\vec{F}(t)-\vec{f}(t)\right| > \epsilon\right) \le e^{-\Omega(n\epsilon^2)}.$$

Thus, the approximation (1) is valid; the f_i's are extremely accurate approximations of the F_i's. This is a consequence of an extremely general mathematical result due to Kurtz [10], [11], [17], which justifies not only the discussion here, but also all of the other differential equation approximations that we make in this work. This approach is typically referred to as taking the *fluid limit* of the system, or as the *mean-field method*.

Of course, we are glossing over some technical details in this discussion. For example, this form of Kurtz's Theorem requires that the items arrive according to a Poisson process, but this can be dealt with by an application of an appropriate Chernoff bound for Poisson random variables. Also, one must formally check that the conditions of Kurtz's Theorem are satisfied for this system. This is straightforward, as the hypotheses of the theorem are very general.

Returning to the analysis of the standard MHT, the differential equation approximation immediately predicts the fraction of the n items that overflow into L; it is just $w \triangleq 1 - \sum_{i=1}^{d} \alpha_i f_i(1)$. While w is not likely to have a simple form, we can easily compute it numerically using standard mathematical software. Furthermore, thinking of w as a function of $\alpha_1, \ldots, \alpha_d$, we can attempt to minimize w subject to the space constraint that $\sum_i \alpha_i \le c$ for some fixed constant c. In other words, we can look for the best sizes for the sub-tables of the MHT, subject to the constraint that the entire MHT has at most c buckets per item that we wish to insert. Since we can easily evaluate w, we can find good α_i's by using standard mathematical software for attempting to minimize a black-box function. Such an optimization approach is only possible because the differential equations are so easy to evaluate. If we were using a less efficient method, such as simulation, the optimization procedure would likely be too slow to be of any real use. We use this technique to configure and compare all of our schemes in Section VII.

IV. Allowing One Move: A Conservative Scheme

We have already argued that the standard MHT is very amenable to a hardware implementation. Our results in Section VII also show that it is quite effective. (This in itself is not really new, given the theoretical analysis of [2] and the subsequent theoretical and numerical results of [9].) Thus, we are now in a position to push the envelope, introducing progressively more complexity into the MHT insertion procedure to reduce the space required while still ensuring that hardware implementations are practical. In particular, as described in the introduction, we are inspired by the existing literature on hashing schemes that allow moves. That inspiration leads us to design alternative MHT insertion procedures that allow a single move per insertion operation, in order to effectively balance performance improvements to the MHT against increased complexity in a hardware implementation. We consider a variety of procedures, in order to gain an understanding of the landscape and how the theory applies.

We start with a fairly conservative scheme. Each bucket of the MHT can be either *marked* or *unmarked*. Initially, all buckets are unmarked. When inserting an item x into the MHT, we find the smallest i such that $T_i[h_i(x)]$ is empty, if there is one. In this case, we simply set $T_i[h_i(x)] = x$. If there is no such i, we find the smallest $j < d$ such that $T_j[h_j(x)]$ is unmarked, if there is one. If there is no such j, we place x on L. If there is such a j, we mark $T_j[h_j(x)]$, set $y = T_j[h_j(x)]$, and look for the smallest $k > j$ such that $T_k[h_k(y)]$ is empty, if there is one. If there is no such k, then we place x on L. If there is such a k, then we set $T_k[h_k(y)] = y$ and $T_j[h_j(x)] = x$.

In words, this scheme tries to place x into a sub-table using the standard MHT scheme. If this fails, it tries to *bump* an item y that it collides with, replacing it and moving y to the right. The item x tries to bump at most one other item. Because of this, we take care to mark an item that we already know cannot be moved to the right successfully to avoid wasting effort. Hence the item that x tries to bump is the leftmost one that we do not know for a fact cannot be bumped.

We say that this scheme is conservative because it is hesitant to move an item. Indeed, the scheme only attempts to move items as the standard MHT insertion procedure breaks down; it makes no attempt to arrange items in advance to prevent this from happening. As such, in practice it is fairly rare for this scheme to actually move an item. We show this in detail in Section VII.

As in Section III, suppose that we start at time 0 with an empty MHT, and insert n items at times $1/n, 2/n, \ldots, 1$. For convenience, we define the notation $[r] = \{1, \ldots, r\}$ for any integer r. For $i \in [d]$, let $F_i(t)$ denote the fraction of buckets in T_i that are occupied at time t, and for $i \in [d-1]$, let $G_i(t)$ denote the fraction of buckets in T_i that are marked. Then a straightforward, if somewhat tedious, probability calculation tells us that the F_i's and G_i's can be approximated by the solution of the following system of differential equations (with $f_i(0) = 0$ and $g_i(0) = 0$).

$$\begin{aligned}\frac{\mathrm{d}f_i}{\mathrm{d}t} &= \frac{1-f_i}{\alpha_i} \\ &\quad \times \left[\prod_{j=1}^{i-1} f_j + \sum_{j=1}^{i-1}\left(\prod_{k=1}^{d}\begin{cases} g_k & k < j \\ f_k - g_k & k = j \\ f_k & k > j\end{cases}\right)\prod_{k=j+1}^{i-1} f_k\right] \\ \frac{\mathrm{d}g_i}{\mathrm{d}t} &= \frac{1}{\alpha_i}\prod_{j=1}^{d}\begin{cases} g_j & j < i \\ f_j - g_j & j = i \\ f_j & j > i\end{cases}\end{aligned}$$

Here we have used an array notation to simplify the presentation of the equations, as the terms of the product are case-dependent on the index. As before, $w \triangleq 1 - \sum_{i=1}^{d} \alpha_i f_i(1)$ is the asymptotic fraction of items that overflow into L.

We can also use the differential equation approach to determine the asymptotic fraction of insertion operations that require an item to be moved in the MHT. Indeed, if $M(t)$ is the fraction of the n items that are inserted at or before time t and required a move in the MHT, then another calculation tells us that $M(t)$ is approximated by the solution $m(t)$ of the differential equation (with $m(0) = 0$)

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \sum_{i=1}^{d-1} \left(1 - \prod_{j=i+1}^{d} f_j \right) \prod_{j=1}^{d} \begin{cases} g_j & j < i \\ f_j - g_j & j = i \\ f_j & j > i \end{cases} .$$

The asymptotic fraction of the n insertion operations that require a move in the MHT is then $m(1)$.

As in Section III, we can now use numerical methods to analyze this scheme in greater detail. We present the results in Section VII.

```
 1: for $i = 1$ to $d - 1$ do
 2:     if $T_i[h_i(x)]$ is not full then
 3:         $T_i[h_i(x)] \leftarrow x$
 4:         return
 5:     $y \leftarrow T_i[h_i(x)]$
 6:     if $T_{i+1}[h_{i+1}(x)]$ is full then
 7:         if $T_{i+1}[h_{i+1}(y)]$ is not full then
 8:             $T_{i+1}[h_{i+1}(y)] \leftarrow y$
 9:             $T_i[h_i(x)] \leftarrow x$
10:             return
11: if $T_d[h_d(x)]$ is not full then
12:     $T_d[h_d(x)] \leftarrow x$
13: else
14:     Add $x$ to $L$
```

Fig. 1. Pseudocode for inserting an item x in the second chance scheme.

V. The Second Chance Scheme

Recall that the scheme in Section IV rarely moves an item in the MHT, even though we have, in principle, allowed ourselves to perform at most one move per insertion operation. Indeed, in many hardware applications, the frequency with which we perform moves may be much less important than the guarantee that we never perform more than one move per insertion. With this in mind, we introduce a scheme that is considerably more aggressive in performing moves, while still guaranteeing that no insertion operation requires more than one.

For intuition, consider inserting n items using the standard MHT insertion scheme. It is fairly clear, both from the definition of the scheme and from the differential equations in Section II, that as the items are inserted the sub-tables fill up from left to right, with newly inserted items cascading from T_i to T_{i+1} with increasing frequency as T_i fills up. Thus, it seems that a good way to reduce the overflow from the MHT is to slow down this cascade at every step.

This idea is the basis for our new scheme, which we call the *second chance* scheme. The formal pseudocode is given in Figure 1. The basic idea is that whenever an inserted item x cannot be placed in T_i, it checks whether it can be inserted into T_{i+1}. If it cannot be placed there then, rather than simply moving on to T_{i+2} as in the standard scheme, the item x checks whether the item y in $T_i[h_i(x)]$ can be moved to $T_{i+1}[h_{i+1}(y)]$. If this move is possible, then we move y and replace it with x. Thus, we effectively get a *second chance* at preventing a cascade from T_{i+1} to T_{i+2}.

From a hardware implementation perspective, this scheme is much more practical than it may first seem. To see this, consider a standard MHT implementation where we can read and hash one item from each sub-table in parallel. In this setting, we can insert an item x using the second chance scheme by reading and hashing all of items in $T_1[h_1(x)], \ldots, T_{d-1}[h_{d-1}(x)]$ in parallel. Once we have the hash values for these items, we can determine exactly how x should be placed in the table using the pseudocode in Figure 1.

Alternatively, if we have a hardware implementation that forces us to read and hash these items sequentially, then we can consider storing hash values for the items in a separate table with sub-tables $T'_1, \ldots, T'_{d-1}$, mimicking the MHT, that can be accessed more quickly. Note that, for an item y in T_i, we only need to store the value of $h_{i+1}(y)$ in $T'_i[h_i(t)]$. If hash values are much smaller than the items themselves (as is often the case), the space needed for the T''s is offset by the reduction in size of the MHT.

As before, we can use differential equations to approximate the fraction of items that overflow into L. In the usual way, suppose that we start at time 0 with an empty MHT, and insert n items into it, at times $1/n, 2/n, \ldots, 1$. For $i \in [d]$, let $F_i(t)$ denote the fraction of buckets in T_i that are occupied at time t. Perhaps more subtly, for $i \in [d-1]$, we let $G_i(t)$ denote the fraction of buckets in T_i at time t that contain an item y such that $T_{i+1}[h_{i+1}(y)]$ has already been found full in line 7 of Figure 1. For such an item y, it is guaranteed that $T_{i+1}[h_{i+1}(y)]$ is occupied at time t and thereafter; since this behavior is different from items where $h_{i+1}(y)$ has never been checked, we must record their growth separately. Another probability calculation now tells us that the F_i's and G_i's can be approximated by the solution of the following system of differential equations (with $f_i(0) = 0$ and $g_i(0) = 0$).

$$\frac{\mathrm{d}f_1}{\mathrm{d}t} = \frac{1 - f_1}{\alpha_1}$$

$$\frac{\mathrm{d}f_i}{\mathrm{d}t} = \frac{1 - f_i}{\alpha_i} \left(f_{i-1} + (f_{i-1} - g_{i-1}) f_i \right) \times \prod_{j=1}^{i-2} g_j + (f_j - g_j) f_{j+1} \qquad \text{for } i = 2, \ldots, d$$

$$\frac{\mathrm{d}g_i}{\mathrm{d}t} = \frac{(f_i - g_i) f_{i+1}}{\alpha_i} \prod_{j=1}^{i-1} g_j + (f_j - g_j) f_{j+1}$$

The asymptotic fraction of items that overflow into L is then just $w \triangleq 1 - \sum_{i=1}^{d} \alpha_i f_i(1)$.

As before, we can also define $M(t)$ to be the fraction of the n items that are inserted at or before time t and require a move in the MHT. In this case, $M(t)$ is approximated by the solution $m(t)$ of the differential equation (with $m(0) = 0$)

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \sum_{i=2}^{d}(f_{i-1} - g_{i-1})f_i(1 - f_i)\prod_{j=1}^{i-2} g_j + (f_j - g_j)f_{j+1}.$$

We analyze this scheme using numerical methods in Section VII.

A. The Extreme Second Chance Scheme

We briefly describe a further enhancement of the second chance scheme, that we call the *extreme second chance scheme*. The idea is fairly intuitive. In the regular second chance scheme, we only consider inserting an item x into T_i if it collides with items $y_1 = T_1[h_1(x)]$,. . ., $y_{i-1} = T_{i-1}[h_{i-1}(y)]$ and for each $j < i - 1$, the bucket $T_{j+1}[h_{j+1}(y_j)]$ is occupied, so that we cannot move y_j to T_{j+1} and place x in T_j. Taking this idea to the extreme, we simply allow for many more possible moves. Thus, in the new scheme, we only consider inserting an item x into T_i if it collides with items $y_1 = T_1[h_1(x)], \ldots, y_{i-1} = T_{i-1}[h_{i-1}(y)]$ and for any $j < k \leq i - 1$, the bucket $T_k[h_k(y_j)]$ is occupied, so that we cannot move y_j to T_k and place x in T_j.

More formally, we have the following specification. Suppose we let $y_i = T_i[h_i(x)]$ if $T_i[h_i(x)]$ is not empty. Let z_0 be the index of the leftmost sub-table that is currently empty for x, with the notation that $z_0 = d + 1$ if none of the d choices are available. For each y_i, let z_i be the index of the currently leftmost empty sub-table for y_i. If $z_0 \leq \min_i z_i$, then we place x in the leftmost sub-table with an empty slot for x. Otherwise, let y_j be the item with the smallest value of z_j, breaking ties in favor of the smallest index j. We move y_j to the leftmost sub-table with an empty slot for y_j and put x in its place. Of course, if we cannot insert x using this procedure, then we place x on L.

There are a few negative aspects of this scheme. First, it requires significantly more hashes and other computations than the original second chance scheme. This may or may not be important, depending on the context. Arguably, the gain in space could conceivably be worth the additional complexity. Second, from our perspective, another clear negative is that while our differential equation analysis could be applied to this scheme, the resulting explication of cases leads to a very large and unwieldy set of equations. For these reasons, we defer a more detailed analysis of it to future work and focus our analysis on the original second chance scheme.

VI. Multiple Items Per Bucket

We can also consider schemes that allow multiple items to be stored in a bucket of the MHT. We do this primarily to illustrate the generality of our methodology, and we emphasize that everything that we do in this section is by way of example; the overall approach can be adapted to an enormous variety of different settings and schemes.

We focus on the case where a bucket in the MHT can store at most two items, and we consider a natural modification of the second chance scheme. (The ideas can be extended to larger numbers of items per bucket.) We can simply modify the procedure in Figure 1 so that when we try to insert an item x into T_i and the buckets $T_i[h_i(x)]$ and $T_i[h_{i+1}(x)]$ are full, we check both $y \in T_i[h_i(x)]$ to see whether either can be moved to $T_{i+1}[h_{i+1}(y)]$, and in this case, we perform the move and place x in $T_i[h_i(x)]$. Of course, we need to consider the case where both $y \in T_i[h_i(x)]$ can be moved, since then we have to choose between them. For simplicity, we suppose that we move the first y that we check that can be moved, so that we do not even have to check the other one. Of course, one could also consider the natural variation where we try to minimize the number of items in $T_{i+1}[h_{i+1}(y)]$. As before, this is only an example, and so we focus on our slightly simpler scheme.

We can use differential equations to approximate the fraction of items that overflow into L. As before, suppose that we start at time 0 with an empty MHT, and insert n items into it, at times $1/n, 2/n, \ldots, 1$. For $i \in [d]$ and $j \in \{0, 1, 2\}$, let $F_{i,j}(t)$ denote the fraction of buckets in T_i that are occupied at time t. For $i \in [d - 1]$ and $j \in \{0, 1, 2\}$, let $G_{i,j}(t)$ denote the fraction of buckets in T_i at time t that contain two items, j of which are items y such that $T_{i+1}[h_{i+1}(y)]$ has already been found full in the equivalent of line 7 in Figure 1; for such an item y, it is guaranteed that $T_{i+1}[h_{i+1}(y)]$ is full at time t and thereafter. Another straightforward probability calculation now tells us that the $F_{i,j}$'s and $G_{i,j}$'s can be approximated using the solution of the following system of differential equations (with $f_{i,j}(0) = \mathbf{1}\,(j = 0)$ and $g_{i,j}(0) = 0$, where $\mathbf{1}\,(\cdot)$ denotes the indicator function).

$$f_{i,\geq j} = \sum_{k=j}^{2} f_{i,k} \qquad g_{i,\geq j} = \sum_{k=j}^{2} g_{i,k} \qquad \text{for } j = 1, 2$$

$$f_{i,0} = 1 - f_{i,\geq 1} \qquad g_{i,0} = f_{i,2} - g_{i,\geq 1}$$

$$\begin{aligned}
\frac{\mathrm{d}f_{1,\geq j}}{\mathrm{d}t} &= \frac{f_{1,j-1}}{\alpha_1} \\
\frac{\mathrm{d}f_{i,\geq j}}{\mathrm{d}t} &= \frac{1}{\alpha_i}\left[\prod_{r=1}^{i-2}\sum_{s=0}^{2} g_{r,s} f_{r+1,2}^{2-s}\right] \\
&\qquad \times \sum_{r=0}^{2} g_{i-1,r} \sum_{s=0}^{2-r} f_{i,2}^{s} f_{i,j-1} \quad \text{for } 2 \leq i \leq d \\
\frac{\mathrm{d}g_{i,\geq j}}{\mathrm{d}t} &= \frac{1}{\alpha_i}\left[\prod_{r=1}^{i-1}\sum_{s=0}^{2} g_{r,s} f_{r+1,2}^{2-s}\right]\left[\sum_{r=0}^{j-1} g_{i,r} f_{i+1,2}^{j-r}\right]
\end{aligned}$$

In this setting, the asymptotic fraction of items that overflow into L is $w \triangleq 1 - \sum_{i=1}^{d}\sum_{j=1}^{2} j\alpha_i f_{i,j}(1)$. (Buckets containing two items must be multiplied by a factor of two.)

If $M(t)$ is again the fraction of the n items that are inserted at or before time t and require a move in the MHT, then $M(t)$ is approximated by the solution $m(t)$ of the differential equation (with $m(0) = 0$)

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \sum_{i=2}^{d}\left[\prod_{r=1}^{i-2}\sum_{s=0}^{2} g_{r,s} f_{r+1,2}^{2-s}\right] f_{i,2} \sum_{r=0}^{1} g_{i-1,r}\left(1 - f_{i,2}^{2-r}\right).$$

VII. Evaluation (Insertions Only)

In this section, we use the differential equations previously derived to assess and compare the performance of the four schemes we consider: the standard MHT scheme (Std), the conservative scheme (Cons) of Section IV, the second chance scheme (SC), and the modification of the second chance scheme for the case of two items per bucket (SC2) discussed in Section VI. For comparison, we also occasionally consider the extreme second chance scheme (SCExt). For that scheme, however, we use simulations, since the corresponding system of differential equations is too unwieldy to present formally in this paper (although such equations could certainly be derived).

As mentioned previously, we solve all differential equations numerically using standard mathematical software. Similarly, we use standard numerical optimization procedures to compare schemes in the following way: for each scheme and a particular d and bound on the number of buckets per inserted item in the MHT, we choose the α_i's in an attempt to minimize w subject to the space constraint. Specifically, we use the NDSolve and NMinimize functions in Mathematica 5.2, with occasional minor modifications to the default behavior of NMinimize (such as trying multiple values for the Method and number of iteration parameters when appropriate). We also verify all of our numerical results through simulation.

For some perspective, the total computation time for all of the results presented here was about a few days on a standard workstation PC obtained in 2004. The coding time was also quite small. Indeed, the total time spent writing and executing all the necessary code for this entire paper was at most two or three weeks for the graduate student author, and that includes the learning curve on Mathematica. We point out these facts to give evidence that this methodology is very practical and, as we shall see shortly, also quite effective.

In what follows, we use the notation that the space of the hash table excluding the CAM is equal to the size of cn items, where c is a constant independent of n. For every scheme presented except the SC2 scheme, this is the same as saying that there are cn buckets; because the SC2 scheme holds up to two items per bucket, the total number of buckets for the SC2 scheme is $cn/2$.

We start by comparing the overflow rates w of the different schemes for different values of c when $d = 4$. We focus on the choice of $d = 4$ because it achieves an excellent tradeoff performance and practicality for a hardware implementation for a reasonable range of values of n. We also focus on values of c that provide the big picture of how the schemes compare. As we have suggested, further experimentation for specific cases (different values of c, d, or different target values for w) would be simple to compute.

The results are displayed in Figure 2. For completeness, we present simulated results for the SCExt scheme using the α_i's determined during the optimization of the SC scheme. The simulated values are determined through our standard simulation methodology, described in detail below. As is clear from the figure, the overflow rate drops off exponentially for all schemes as c increases. Also, each increase in complexity to the MHT insertion scheme gives a significant reduction in the resulting overflow rate. This difference is profound for the SC2 scheme; the additional flexibility from having two items to move at each level offers substantial benefits, at the cost of some hardware complexity and the requirement that two items can be stored in a single bucket. More interestingly, though, the differences between the Std, Cons, and SC schemes are enormous as c grows towards 2, and the difference is still significant even for c much closer to 1 (say, $c = 1.2$).

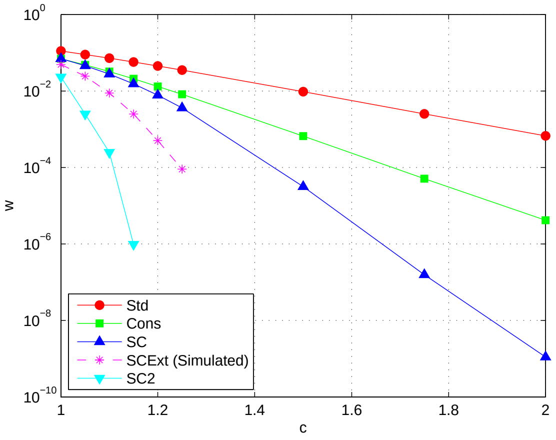

Fig. 2. Comparison between the asymptotic fractional overflows w and the corresponding space multiplier c for $d = 4$.

To get a more refined comparison of the schemes, we fix a *target* value of $w = 0.2\%$, and attempt to find the smallest c for each scheme that achieves w using NMinimize, taking c to the nearest hundredth. Roughly speaking, this corresponds to drawing the horizontal line $w = 0.2\%$ in Figure 2 and comparing the c coordinates where that line intersects each of the lines corresponding to the different schemes. The value for w is chosen so that for $n = 10000$, which we consider a reasonable size, the expected number items that overflow is about 20. As we shall see later, the distribution of the number of items that overflow in this case is approximately $\mathrm{Poisson}(20)$. The probability that a $\mathrm{Poisson}(20)$ random variable exceeds, say, 64 is approximately (by numerical calculation) 3.77×10^{-15}, which is negligible. Thus for these values, we can, in practice, just use a CAM of size 64 to represent the overflow list L.

We present the results of our comparison of the schemes for the target $w = 0.2\%$ for $d \in \{3, 4, 5\}$ in Table I. In that table, we also show the asymptotic fraction m of the n insertion operations that require a move, as well as the values for the α_i's determined by our optimizations. As mentioned previously, we are principally interested in the results for $d = 4$. The results for the other values for d are of secondary importance and are presented mostly for the sake of comparison with the results for $d = 4$.

Table I essentially confirms our original intuition concerning the schemes and the overall picture suggested by Figure 2.

TABLE I
SCHEME COMPARISON FOR A TARGET $w = 0.2\%$

Scheme	c	m	α_1	α_2	α_3
Std	2.67	0%	1.4004	0.8373	0.4616
Cons	1.75	1.84%	0.7743	0.6048	0.3740
SC	1.62	8.54%	0.7121	0.6385	0.2705
SC2	1.22	11.9%	0.2062	0.2031	0.2016

(a) $d = 3$

Scheme	c	m	α_1	α_2	α_3	α_4
Std	1.79	0%	0.7856	0.5143	0.3150	0.1781
Cons	1.39	1.6%	0.5226	0.4140	0.2804	0.1775
SC	1.29	12.0%	0.4695	0.4563	0.2512	0.1082
SC2	1.06	14.9%	0.1997	0.1983	0.1008	0.0273

(b) $d = 4$

Scheme	c	m	α_1	α_2	α_3	α_4	α_5
Std	1.39	0%	0.536	0.335	0.253	0.158	0.110
Cons	1.24	1.45%	0.393	0.335	0.243	0.162	0.108
SC	1.16	15.2%	0.367	0.367	0.239	0.129	0.058
SC2	1.02	18.5%	0.103	0.103	0.103	0.102	0.101

(c) $d = 5$

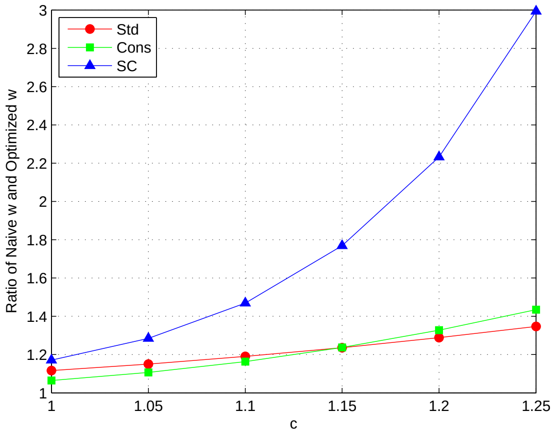

Fig. 3. Effect on w of optimizing the α_i's for $d = 4$ and various c values.

TABLE II
EFFECT OF OPTIMIZING THE α_i'S FOR $d = 4$ AND TARGET $w = 0.2\%$

Scheme	Naive c	Optimized c	Ratio
Std	2.00	1.79	1.12
Cons	1.46	1.39	1.05
SC	1.41	1.29	1.09
SC2	1.14	1.06	1.08

In all cases, Cons gives a significant reduction in space over Std, at the cost of performing moves during a small fraction of insertion operations. SC2 provides a significant further improvement over the space requirement of Cons, at the cost of a (likely reasonable) order of magnitude increase in m. (As indicated in Figure 2, this improvement would likely be more dramatic if we considered smaller w, say $w = 2 \times 10^{-4}$, which would correspond to our current setup with $n = 10^5$ instead of the current value $n = 10^4$.) Finally, SC2 gives an even further reduction in the space requirement with a further small increase in the frequency of moves. Note that SC2 is slightly more complex, and requires the item size be such that two items can naturally fit into a bucket. Furthermore, as we promised earlier, $d = 4$ gives an excellent tradeoff between the number of hash functions and the performance of the schemes. The space requirements are much worse for $d = 3$, and may not be sufficiently better for $d = 5$ to justify the use of another hash function and accompanying sub-table (assuming the use of Cons, SC, or SC2).

Of course, while our previous results rely on optimizing the α_i's, it is likely impractical to use those exact values in a real application. Nevertheless, it is still worthwhile to try to optimize the α_i's within the domain of practicality. To give a rough illustration of this, we first compute the values of w corresponding to the various schemes and values of c depicted in Figure 2 under the naive configuration where $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = c/4$. We then compare these values of w to the optimized values shown in Figure 2, and plot a subset of them that is easy to visualize in Figure 3. For the Std, Cons, and SC schemes, the trends are clear. All of these schemes benefit significantly from optimization, although the SC scheme clearly benefits much more than the others. Although we have not plotted it, the situation for SC2 is even more extreme; the ratio for $c = 1$ is 1.66, and by $c = 1.15$ it grows to 1457. Also, we perform a comparable simulation for the SCExt scheme, comparing the values from Figure 2 with an estimate of the overflow rate when all sub-tables have equal size, based on simulations. In this case, the ratio for $c = 1$ is 1.03, and by $c = 1.25$ it grows to 8.58.

To get a more refined look at the effect of optimizing the α_i's for the various schemes, we once again consider the target value $w = 0.2\%$. We compare the optimized c values from Table I(b) with their corresponding naive values, which are obtained by searching for the smallest value of c that achieves the target $w = 0.2\%$ under the assumption that all of the α_i's are equal. The results are illustrated in Table II. We see that the savings in space from optimizing the α_i's is on the order of 10%; it would be slightly higher for smaller values of w. We conclude that optimizing the α_i's is likely to be worthwhile in practice, if the design naturally allows varying sub-table sizes.

Finally, as our results arise from the fluid limit approximation, it is important to verify them independently through simulation to ensure that they are accurate. (The optimization of the α_i's are purely numerical; it cannot be efficiently performed through simulation.) We verify all of the results in Figure 2 and Tables I and II in the following way. For a given scheme and $\alpha_1, \ldots, \alpha_d$ we simulate the insertion of $n = 10000$ items into an MHT with d sub-tables, where the size of T_i is $\lfloor \alpha_i n \rfloor$. (We round down for simplicity; the tables are large enough that losing a bucket makes an insubstantial difference.) We keep track of the fraction of items placed in the overflow list L and the fraction of insertion operations resulting in a move, and average the results over 10^5 trials. We sample all hash values using the standard Java pseudorandom number generator.

The simulation results indicate that the differential approximations are highly accurate. For the Std and Cons schemes, the largest relative error for the simulated overflow rates measured against the numerically calculated rates is 0.9% for calculated rates more than 10^{-4}, only 1.8% for calculated rates more than 10^{-6}, and 170% for the remaining rates (which is fairly accurate, considering that the remaining calculated rates are so small compared to the number of trials in the experiment). Similarly, the largest relative error in the fraction of insertions requiring a move is 0.12%. For the SC2 scheme, the situation is more complicated, most likely due to the complexity of the differential equations. The differential equation approximation seems to be very accurate for the first few sub-tables of the MHT, but the relative error degrades to about 22% for the overflow list. Running the simulation again for $n = 10^5$ helps immensely. (Due to time constraints, we only try $n = 10^5$ for the data points in Figure 2 and Table I.) In this case, the relative errors in the overflow rates are less than 1% when the calculated overflow rate is at least 10^{-6}. The relative errors in the fractions of insertions requiring a move are on the order of 10^{-5}.

Having examined the overflow rate w for the various schemes, we now look at some of the finer properties of the distribution of the items in the T_i's. Suppose that the differential equations for a scheme tell us that the fraction of the n items inserted into Q is f, where Q is some sub-table of the MHT or L. If the events that each of n items ends up in Q were independent with probability f, then the number of items in Q would be $\text{Binomial}(n, f)$. If f were on the order of λ/n for some small constant λ, then the distribution $\text{Binomial}(n, f)$ would be approximately $\text{Poisson}(\lambda)$. For larger f, we would expect the distribution to be normal around its mean by the central limit theorem.

Obviously, each item does not get inserted into Q with the same probability f; the probabilities change according to the differential equations. However, under the heuristic assumption that the events at each step are independent with the appropriate probabilities, it is straightforward to generalize the standard generating function proof for the Binomial convergence to the Poisson distribution (e.g. [8, Exercise 5.12.39a]). Hence, if f is on the order of λ/n for some small constant λ, then we expect the distribution of the number of items in Q to be nearly $\text{Poisson}(\lambda)$. Similarly, if f is much larger, then the distribution of the number of items in Q should be approximately normal. Thus, we expect that for a well-configured scheme, the distribution of the number of items in each sub-table of the MHT should be approximately normal, and the number of items in the overflow list L should be approximately Poisson. In fact, the conclusion that the size of L is approximately Poisson is absolutely critical for designing a practical system, so that we can determine the size of the CAM we need. Indeed, recall that the way we choose the target probability $w = 0.2\%$ for $n = 10000$ in Table I is entirely dependent on such reasoning.

We test the accuracy of this intuition through simulation. For each of the configurations in Table I(b), we run 10^5 trials of the experiment previously outlined, and record, for each trial, the distribution of the $n = 10000$ items over the sub-tables and L. We use this data to test our intuition that the number of items in a particular sub-table is approximately normally distributed, and that the number of items in L is approximately Poisson.

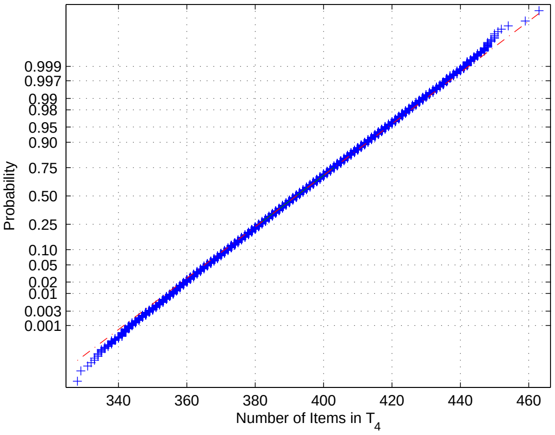

Fig. 4. Normal probability plot of the number of items in T_4 when simulating the second chance scheme with $d = 4$, $n = 10000$, and target $w = 0.2\%$.

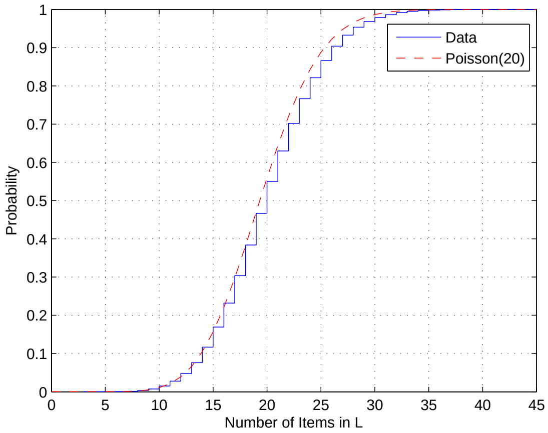

Fig. 5. Empirical cumulative distribution function of the number of items in L when simulating the second chance scheme with $d = 4$, $n = 10000$, and target $w = 0.2\%$, as compared with the cumulative distribution function of a $\text{Poisson}(20)$ random variable.

We focus on the second chance scheme, as the results for the other schemes are similar. (The approximations are not quite as good for the SC2 scheme, however, but this is most likely caused by the same rate of convergence issue encountered earlier in our simulation results.) In Figure 4, we give a normal probability plot of the number of items in T_4 (the plots for the other sub-tables are similar). Intuitively, the data is plotted so that if the samples were taken independently from a normal distribution, then all of the points would form an approximately straight line with overwhelming probability. Looking at the plot, the data distribution clearly appears to be very close to a normal distribution. Similarly, in Figure 5 we plot the empirical cumulative distribution function of the

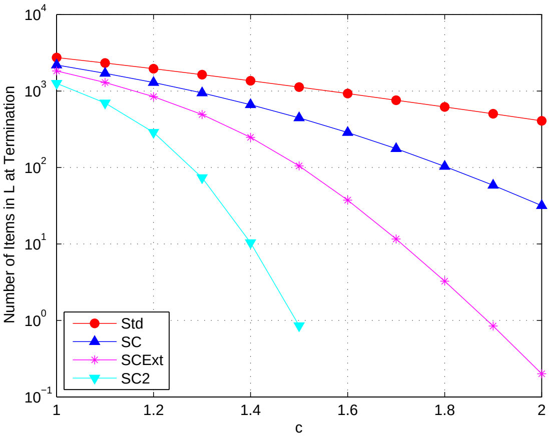

Fig. 6. Number of items in L after inserting 10,000 items into 4 equal-sized tables, followed 100,000 alternations of deletions and insertions.

number of items in L and, for comparison, the cumulative distribution function of the $\text{Poisson}(20)$ distribution. The similarity is immediate.

VIII. Deletions

We briefly discuss the implementation and evaluation of our schemes when deletions are allowed. We emphasize that the discussion here is preliminary, and a more complete analysis will appear in future work.

Most of the schemes that we consider in this paper naturally support deletions. In particular, for the standard MHT insertion procedure and the second chance scheme and its variants, we can delete an item simply by removing it from the MHT. The situation for the conservative scheme is more complicated, as the markings on the buckets must be occasionally updated. For simplicity, we omit that scheme from our discussion.

To get a sense of how deletions can affect our schemes, we perform a very simple simulation. We consider inserting 10,000 items into an empty MHT with 4 equal-sized sub-tables, and then performing 100,000 alternations between deleting a random item in the table and inserting a new item. We average the results over 10,000 trials and show the results in Figure 6. It is clear that our schemes provide a significant advantage over the standard scheme, although the improvement is less substantial than in the case of insertions only. (The equilibrium distribution of the fullness of the various sub-tables is much different when deletions occur.) In future work, we will present fluid limit descriptions of this deletion model, as well as an arguably more realistic model where we consider the lifetime distribution of an item in the table.

IX. Conclusion

We have shown that it is possible and practical to significantly increase the space utilization of a multiple choice hashing scheme by allowing a single move during an insertion procedure. Furthermore, our efforts bridge the theory and practice of such schemes, and provide a solid methodology for future work.

Acknowledgments

The authors were supported in part by grants from Cisco Systems and Yahoo! Research. Adam Kirsch was also supported by an NSF Graduate Research Fellowship.

References

[1] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal of Computing* 29(1):180-200, 1999.

[2] A. Broder and A. Karlin. Multilevel adaptive hashing. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 43-53, 1990.

[3] A. Broder and M. Mitzenmacher. Using multiple hash functions to improve IP Lookups. In *Proceedings of IEEE INFOCOM*, pp. 1454-1463, 2001.

[4] J. Byers, J. Considine, and M. Mitzenmacher Geometric generalization of the power of two choices. In *Proceedings of the 16th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pp. 54-63, 2004.

[5] M. Dietzfelbinger and C. Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380:(1-2):47-68, 2007.

[6] C. Estan, K. Keys, D. Moore, and G. Varghese. In *Proceedings of the 2004 SIGCOMM*, pp. 245–256, 2004.

[7] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space Efficient Hash Tables With Worst Case Constant Access Time. *Theory of Computing Systems*, 38(2):229-248, 2005.

[8] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Third Edition, Oxford University Press, 2003.

[9] A. Kirsch and M. Mitzenmacher. Simple Summaries for Hashing with Choices. *IEEE/ACM Transactions on Networking*, to appear. Temporary version available at: http://www.eecs.harvard.edu/~kirsch/pubs/sshmc/ton-cr.pdf.

[10] T. G. Kurtz. Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes. *Journal of Applied Probability*, (7):49-58, 1970.

[11] T. G. Kurtz, *Approximation of Population Processes*, SIAM, 1981.

[12] M. Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. Ph.D. thesis, University of California, Berkeley, 1996.

[13] M. Mitzenmacher, A. Richa, and R. Sitaraman. *The Power of Two Choices: A Survey of Techniques and Results*, edited by P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim. Kluwer Academic Publishers, Norwell, MA, 2001, pp. 255-312.

[14] M. Mitzenmacher and B. Vöcking. The Asympotics of Selecting the Shortest of Two, Improved. In *Analytic Methods in Applied Probability: In Memory of Fridrikh Karpelevich*, American Mathematical Society, pp. 165-176, 2003. Edited by Y. Suhov.

[15] A. Pagh and F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122-144, 2004.

[16] R. Panigrahy. Efficient hashing with lookups in two memory accesses. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 830–839, 2005.

[17] A. Shwartz and A. Weiss. *Large Deviations for Performance Analysis: Queues, Communications, and Computing*. Chapman & Hall, 1995.

[18] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast hash table lookup using extended Bloom filter: an aid to network processing. In *Proceedings of ACM SIGCOMM*, pp. 181-192, 2005.

[19] B. Vöcking. How Asymmetry Helps Load Balancing. *Journal of the ACM*, 50:4, pp. 568-589, 2003.

[20] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed IP routing lookups. *ACM SIGCOMM Computer Communication Review*, 27:4, pp. 25–36, 1997.