Energy-Constrained Balancing

Yossi Kanizo, David Hay, and Isaac Keslassy

***Abstract*—This paper defines and analyzes a fundamental energy-constrained balancing problem, in which elements need to be balanced across resources in order to minimize the increasing convex cost function associated with the load at each resource. However, the balancing operation needs to satisfy average and instantaneous constraints on the energy associated with checking the current load of the many resources.**

In the paper, we first show tight lower and upper bounds on the solution of the problem depending on the specific system parameters. Then, we explain how these solutions can be applied to construct hash tables with optimal variance of the bin size, as well as energy-efficient Bloom filters.

I. Introduction

A. Motivation

Power consumption of contemporary network devices has become a major bottleneck in recent years, due to the continuously-growing demand both in the Internet core and in large datacenters. The large power consumption imposes major energy-expensive cooling mechanisms to prevent the heat from affecting the electronic components. As a result, *energy represents a considerable cost factor in contemporary networks*, and this cost keeps increasing rapidly. For example, power and cooling costs have already become the second-largest contributor to the total cost of datacenters [1]. Moreover, the power and cooling needs are the major reason causing contemporary datacenter facilities to reach their full capacity, and therefore to build and migrate to newer facilities. Finally, renewed environmental concerns are expected to result in new rules and laws in the near future that impose a *green networking*, i.e. networking with reduced power consumption.

A promising approach to deal with the energy bottleneck is to devise *energy-efficient algorithms and data structures*. When studying high-speed data structures implemented in networking devices, researchers have traditionally been concerned about their worst-case performance, as well as parallelism and pipelining abilities. However, power consumption is essentially independent of these implementation choices and is mainly determined by the *total* (or equivalently, *average*) performance rather than the worst-case per-operation performance.

This paper takes this approach and focuses on the *balancing problem*—a fundamental problem that lies at the core of many operations and applications in modern computer networks, such as routing, switching, packet classification, storage and many more. The balancing problem is also crucial for devising other data structures often used at wire speed, such as hash tables and Bloom filters. Given its applicability, this problem was investigated in many contexts and under many assumptions. Our paper is unique by considering simultaneously both the energy efficiency of the solution (that is, the total number of its operations) and its quality under a large class of targeted utility functions.

We model the balancing problem using the *balls and bins model* [2], and more specifically its *sequential multiple-choice* variant [3]. In this model, n balls are placed in m bins. Before placing a ball, d bins are chosen according to some distribution (e.g., uniformly at random) and the ball is placed in one of these bins following some rule (for example, in the least occupied bin). Note that the process of choosing the bins randomly is equivalent to applying *fully-random* hash functions on the balls. Moreover, we consider an extension of this model that allows a small fraction of the balls not to be placed in the bins; these balls are either disregarded or stored in a dedicated overflow list, usually implemented in an expensive memory (a similar model was considered, for example, in [4]). The quality of the balancing is measured by the load on the bins: The resulting load at each bin induces a certain *cost*, which is calculated by an arbitrary non-decreasing convex *block cost function* ϕ_B. Our goal is naturally to minimize the overall expected cost of the system.

To deal with the power consumption of the insertion algorithm we impose the following restriction: each operation can look at up to $a < d$ bins *on average*, before deciding where to place the ball. Note that in most reasonable scenarios, checking the status of a bin (e.g., its occupancy) corresponds to either a memory access or a probe over the network. Thus, our restriction can be viewed as imposing an *energy budget* on the insertion algorithm. Given this energy budget, we aim at achieving the highest-quality (that is, lowest-cost) balancing.

B. Background

Balancing problems were extensively investigated in the last decades for various applications involving allocations of resources [5]. Prime examples are task balancing between many machines [6], [7], item distribution over several locations [8], bandwidth allocation in communication channels [5] or within switches and routers [9], and hash-based data structures [10].

Our paper is most related to the *sequential static multiple-choice balls-and-bins problem* described above. This model was first considered in the seminal work of Azar, Broder, Karlin, and Upfal [3], and had a large impact on modern algorithms and data structures (see surveys in [11], [12]). Note that most of the papers considered the *maximum load* of the system, while our paper considers the entire load distribution.

Energy-constrained hash-schemes were also considered in [13]. Our paper is different since it deals with a general

This work was partly supported by the European Research Council Starting Grant no. 210389.

Y. Kanizo is with the Dept. of Computer Science, Technion, Haifa, Israel. Email: ykanizo@cs.technion.ac.il.

D. Hay is with the Dept. of Electronics, Politecnico di Torino, Turin, Italy. Email: hay@tlc.polito.it.

I. Keslassy is with the Dept. of Electrical Engineering, Technion, Haifa, Israel. Email: isaac@ee.technion.ac.il.

balancing problem and aims to minimize a general cost function given a known overflow list size, while [13] considered the size of the overflow list given bounded-size bins.

As described later, one of the applications of the balancing problem is the construction of *energy-efficient Bloom-filters*, which also use multiple hash functions. One proposal in this direction is to use a Blocked Bloom Filter [14] in which for each element all hash functions are mapped into a single block in the memory. Although this technique is clearly energy efficient, it suffers from poor performance (e.g. high false positive rate) due to an *imbalance* between the memory blocks. Our paper shows a solution to this problem with significantly better performance.

C. Our Contributions

This paper explores the *optimality region* of the balancing problem.

We first provide lower bounds on the minimum cost of each instance of the problem. The lower bound depends on the energy budget a, the number of hash functions d, and the overflow list size, but does not depend on the block cost function ϕ_B. Our lower bounds hold when all hash functions have uniform distribution or when their overall distribution is uniform (in the latter case, the hash function distributions can be different).

Then, we provide three different schemes that meet the lower bounds on different energy budgets; we further find the minimum size of the overflow list that should be provided in order to achieve optimality. All our analytical models are compared with simulations showing their accuracy.

We conclude by showing how, with a careful choice of the block cost function ϕ_B, the balancing problem can be directly used to optimize two important data structures, which are widely-used in wire-speed algorithms:

First, we consider a *hash table* in which random hash functions map elements to associated buckets, and *chaining* is used to solve hash collisions: if two items are mapped to the same bucket they will be stored in a corresponding *linked list*. Notice that the longer the linked list is, the longer it takes to query the bucket. Many studies investigated the performance of such hash tables, focusing mainly on the worst-case query-time (see [15] for further discussion). In this paper we show how solving the balancing problem with a quadratic block cost function can be used in order to devise such hashing schemes with optimal *variance* as well as energy efficiency.

The second application we analyze deals with *Bloom filters*, which are space-efficient randomized data structures that support approximate set membership queries [16]. The quality of a Bloom filter is measured by its *false positive rate*, i.e. the probability that a set membership query returns TRUE, while the element is not in the set; Bloom filters always have *zero false negative rate*. Bloom Filters are often used in network applications [17] especially when the set is very large, when the memory is scarce (e.g. high-speed on-chip memory), or when it should be shared across many nodes in a limited-bandwidth network [18]. Usually Bloom Filters are not energy-optimized and do not take into account the structure of their underlying memory (namely, the memory block size). We will show how to use the balancing problem with a specific block cost function in order to build Bloom filters that have low false positive rate and consume significantly less power than the traditional architecture.

Paper Organization: The optimal balancing problem is defined in Section II, followed by our lower bound results in Section III. The three optimal schemes and their analysis are presented in Sections IV, V, and VI, while a comparative study appears in Section VII. Two applications of the balancing problems, namely chain-based hash tables and Bloom filters, are presented in Sections VIII and IX, respectively. Finally, Section X gives concluding remarks.

II. Problem Statement

In this section, we define the notations and settings of the *optimal balancing problem*.

More specifically, let $\mathcal{B}$ be a set of m *buckets* of unbounded size (also referred to as *bins* or *blocks*) and let $\mathcal{E}$ be a set of n *elements* (or *balls*) that should be distributed among the buckets. In addition, denote by $r = \frac{n}{m}$ *the element-per-bucket ratio*.

Assume also that there exists an *overflow list* [4], i.e. a special bucket of bounded size $\gamma \cdot n$ (namely, at most a fraction γ of the elements can be placed in the list), which can be used by the insertion algorithm at any time. For example, depending on the application, the overflow list may correspond to a dedicated memory (e.g. CAM) in hardware-implemented hash-table, or to the loss ratio when the hashing scheme is allowed to drop elements.

Elements are inserted into either one of the m buckets or the overflow list, according to some hashing scheme with at most d hashes per element, which is defined as follows [13]:

Definition 1: A *hashing scheme* consists of defining:
(i) d hash-function probability distributions over bucket set $\mathcal{B}$, used to generate a *hash-function set* $\mathcal{H} = \{H_1, \ldots, H_d\}$ of d independent random hash functions;
(ii) and an *insertion algorithm* that places each element $x \in \mathcal{E}$ in one of the d buckets $\{H_1(x), \ldots, H_d(x)\}$ or in the overflow list. The insertion algorithm is an *online* algorithm, which places the elements one after the other with no knowledge of future elements.

The power consumption of a hashing scheme is measured by the number of *bucket accesses* needed to store the incoming elements. We assume that a hashing scheme needs to access a bucket to obtain any information on it. We do not count accesses to the overflow list.

We further consider two constraints, which can be seen as either power- or throughput-constraints depending on the application. First, we require that the *average* number of bucket accesses per element insertion must be bounded by some constant $a \geq 0$. In addition, notice that the *worst-case* number of bucket accesses per element insertion is always bounded by d, because an element does not need to consider any of its d hash functions more than once. These two constraints are captured by the following definition:

Definition 2: An $\langle a, d, r \rangle$ hashing scheme is a hashing scheme that inserts all elements with an average (respectively, maximum) number of bucket accesses per insertion of at most a (respectively, d), when given an element per bucket ratio r.

We are now ready to define the *optimal balancing problem*, which is the focus of this paper. Let $\phi_B : \mathbb{N} \mapsto \mathbb{R}$ be the *block cost function* mapping the occupancy of a bucket to its real-valued cost. We assume that ϕ_B is *non-decreasing* and *convex*. Our goal is to minimize the expected overall cost:

Definition 3: Let O_j be a random variable that counts the number of elements in the j-th bucket. Given γ, a, d and r, the OPTIMAL BALANCING PROBLEM consists in finding an $\langle a, d, r\rangle$ hashing scheme that minimizes

$$\phi^{\text{BAL}} = \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} E\left(\phi_B\left(O_j\right)\right).$$

Whenever defined, let $\phi^{\text{BAL}}_{\text{OPT}}$ denote this optimal expected limit balancing cost.

Note for example that given an identity cost function $\phi_B(x) = x$ and no overflow list ($\gamma = 0$), $\phi^{\text{BAL}}_{\text{OPT}}$ corresponds to the average load per bucket, which is exactly r, no matter what insertion algorithm or hash functions are used.

III. Lower Bounds

A. Uniform Hash Function Distributions

In this section, we show a lower bound on the achievable value of $\phi^{\text{BAL}}_{\text{OPT}}$. This is accomplished when the occupancy of the buckets follows a very particular distribution $P_{\text{LB}}(i)$ that depends on the number of buckets m, the number of elements n, the average number of bucket accesses a, and the overflow fraction γ.

$P_{\text{LB}}(i)$ is derived by computing the best-case distribution of each bucket in an offline setting. In this setting, each memory access is considered as a distinct element, where initially all the $a \cdot n$ distinct elements are hashed to the buckets. Then, $(a-1+\gamma)\cdot n$ are removed in a way that minimizes the cost function ϕ^{BAL} (we end up with exactly $(1-\gamma)\cdot n$ elements in buckets). Since the block cost function ϕ_B is non-decreasing and convex then the marginal cost is the largest in the most occupied buckets. Thus, removing the elements greedily, each time from one of the most occupied buckets, is expected to minimize the lower bound on the achievable value of $\phi^{\text{BAL}}_{\text{OPT}}$. Interestingly, we get that $P_{\text{LB}}(i)$ does not depend on the block cost function ϕ_B.

Theorem 1: Let k_0 be the largest integer such that

$$a \cdot r \cdot \frac{\Gamma\left(k_0, \frac{a\cdot n}{m}\right)}{(k_0-1)!} + k_0 \cdot \left(1 - \frac{\Gamma\left(k_0+1, \frac{a\cdot n}{m}\right)}{k_0!}\right) < r\left(1-\gamma\right),$$

where $\Gamma(s,x) = \int_x^\infty t^{s-1}e^{-t}dt$ is the upper incomplete gamma function. Also, let $e_0 = \frac{a\cdot n}{m}\cdot\Gamma\left(k_0, \frac{a\cdot n}{m}\right)/(k_0-1)!$ and $p_0 = \Gamma\left(k_0+1, \frac{a\cdot n}{m}\right)/(k_0)!$. We define the following distribution $P_{\text{LB}}(i)$ that we name as the lower bound distribution:

$$P_{\text{LB}}(i) = \begin{cases} e^{-\frac{a\cdot n}{m}} \frac{\left(\frac{a\cdot n}{m}\right)^i}{i!} & 0 \le i < k_0 \\ e^{-\frac{a\cdot n}{m}} \frac{\left(\frac{a\cdot n}{m}\right)^{k_0}}{k_0!} + e_0 + k_0 + 1 & i = k_0 \\ -k_0p_0 - p_0 - r\cdot(1-\gamma) & \\ -e_0 - k_0 + k_0p_0 + r\cdot(1-\gamma) & i = k_0+1 \\ 0 & \text{otherwise} \end{cases}$$

Under the constraint that all hash functions are uniform, the optimal expected limit balancing cost $\phi^{\text{BAL}}_{\text{OPT}}$ in the OPTIMAL BALANCING PROBLEM is bounded from below by

$$\phi^{\text{BAL}}_{\text{LB}} = \sum_{j=0}^{k_0+1} P_{\text{LB}}(i)\cdot\phi_B(i).$$

Note that $P_{\text{LB}}(i)$ is independent of the block cost function ϕ_B.

Proof: We derive the lower bound on the balancing by computing the best-case distribution of each bucket in an offline setting. We assume that whenever an hash function points to some bucket, an element is inserted into this bucket, having a total of $a{\cdot}n$ elements at the end of the process. Then, we remove exactly $(a-1+\gamma)\cdot n$ elements, which results in $(1-\gamma)\cdot n$ total elements in the buckets. We remove the elements in a way that minimizes the cost function ϕ^{BAL}.

In fact, by the convexity of the block cost function ϕ_B, minimizing the total cost ϕ^{BAL} can be done by removing the $(a-1+\gamma)\cdot n$ elements greedily, each time from one of the most occupied buckets. This is because the marginal cost is the largest (due to convexity) in those buckets. In the sequel, we relate to this process as the *removal process*.

We consider every hash value as a distinct element. Therefore, the number of elements (out of total $a{\cdot}n$ elements) that are mapped to bucket $j \in \mathcal{B}$ follows a Binomial distribution with $a\cdot n$ independent experiments and a success probability of $\frac{1}{m}$. Let $Q_j(i) = \binom{an}{i}\left(\frac{1}{m}\right)^i\left(1-\frac{1}{m}\right)^{an-i}$ denote the probability that bucket j stores i elements before the removal process.

Let $M_j(i)$ be the probability that bucket j stores i elements *after* the removal process. As we show now, $M_j(i)$ has to satisfy two constraints. First, since in the removal process elements are only removed (and not inserted), then the probability that some bucket stores less than i elements after the removal process cannot be larger than before the removal process. Thus, for every i:

$$\sum_{k=0}^{i} M_j(k) \ge \sum_{k=0}^{i} Q_j(k). \tag{1}$$

Second, since we end up with exactly $(1-\gamma)\cdot n$ elements, then:

$$\sum_{k=1}^{m}\left(\sum_{i=0}^{\infty} i\cdot M_k(i)\right) = (1-\gamma)\cdot n, \tag{2}$$

that is, the expected number of elements in all the buckets after the removal process must be $(1-\gamma)\cdot n$.

As we are looking for a lower bound on the balancing cost, our goal is to pick the bucket distribution that minimizes that cost. Consider bucket j and assume that the expected occupancy after the removal process is E_0. Since all hash functions are uniform, by symmetry E_0 must be at most $\frac{an}{m}$. We construct the following distribution that minimizes the balancing cost of bucket j. The idea is to keep the original probabilities for low values of buffer occupancies, until the point where the expected occupancy E_0 is reached. On this point, we share the remaining probabilities such that we get the exact expected occupancy. Specifically, let k_0 be the largest integer such that

$$\sum_{i=0}^{k_0} i \cdot Q_j(i) + k_0 \cdot \left(1 - \sum_{i=0}^{k_0} Q_j(i)\right) < E_0.$$

That is, k_0 is the buffer occupancy until which we keep the original probability. Let $e_0 = \sum_{i=0}^{k_0} i \cdot Q_j(i)$ and $p_0 = \sum_{i=0}^{k_0} Q_j(i)$. In the sequel, we use e_0 and p_0 to construct the remainder of the distribution, that is, the probability for buffer occupancies k_0 and k_0+1.

We define the following distribution $P_j(i)$:

$$P_j(i) = \begin{cases} Q_j(i) & 0 \le i < k_0 \\ Q_j(i) + e_0 + k_0 + 1 & \\ -k_0 p_0 - p_0 - E_0 & i = k_0 \\ -e_0 - k_0 + k_0 p_0 + E_0 & i = k_0 + 1 \\ 0 & \text{otherwise} \end{cases}$$

$P_j(i)$ satisfies both constraints from Equations (1) and (2). First, since we kept the original probabilities until buffer occupancy k_0, and then shared the remaining probabilities between k_0 and k_0+1, then for every i, $\sum_{k=0}^{i} P_j(k) \ge \sum_{k=0}^{i} Q_j(k)$. Second, let $\widetilde{P}_j(i)$ be the random variable that corresponds to the distribution $P_j(i)$. Then, the expected number of elements in bucket j is:

$$\begin{aligned} \mathbf{E}\left(\widetilde{P}_j(i)\right) &= \sum_{i=0}^{k_0+1} i \cdot P_j(i) \\ &= e_0 - k_0 Q_j(k_0) + k_0 P_j(k_0) \\ &\quad + (k_0+1) P_j(k_0+1) \\ &= E_0 \end{aligned}$$

Thus, $P_j(i)$ satisfies the two constraints.

We now show that it minimizes the cost function, over all distributions that satisfy both constraints. Let $G_j(i)$ be a distribution over the buffer occupancies after the removal process that satisfies both constraints. Let i_0 be the smallest integer such that $G_j(i_0) \ne P_j(i_0)$; if such i_0 does not exist, we are done since $G_j(i)$ coincides with $P_j(i)$. We will show that $G_j(i_0) > P_j(i_0)$. Also, let i_1 be the largest integer such that $G_j(i_1) > P_j(i_1)$.

We now show that if i_0 and i_1 are defined, then $G_j(i_0) > P_j(i_0)$, and $i_1 - i_0 \ge 2$. We distinguish between 3 cases: $i_0 > k_0$, $i_0 = k_0$ and $i_0 < k_0$.

First, in case of $i_0 > k_0$, for every bucket occupancy $i \le k_0$, $G_j(i) = P_j(i)$. Thus, $G_j(i) = P_j(i)$ for every i, as $G_j(i)$ satisfies the second constraint (Equation (2)), implying that i_0 and i_1 are not defined.

In case $i_0 < k_0$, by the first constraint (Equation (1)) and the fact that $P_j(i) = Q_j(i)$ for every $i < i_0$, we get that $G_j(i_0) > P_j(i_0)$. We now show that $i_1 > k_0$, implying that $i_1 - i_0 \ge 2$. Assume on the contrary that $i_1 \le k_0$, then $G_j(k_0+1) \le P_j(k_0+1)$ and for every $i > k_0+1$, $G_j(i) = 0$. Let $\widetilde{G}_j(i)$ be the random variable that corresponds to $G_j(i)$. Since $\mathbf{E}\{\widetilde{G}_j(i)\} = \mathbf{E}\{\widetilde{P}_j(i)\}$ and for any random variable X that takes values in $\mathbb{N}$, $\mathbf{E}(X) = \sum_{\ell=1}^{\infty} \Pr\{X \ge \ell\}$, we get that

$$\sum_{i=1}^{k_0+1} \sum_{\ell=i}^{\infty} G_j(\ell) = \sum_{i=1}^{k_0+1} \sum_{k=i}^{\infty} P_j(\ell).$$

Thus,

$$\sum_{i=1}^{k_0+1} \left[\sum_{\ell=i}^{\infty} G_j(\ell) - \sum_{\ell=i}^{\infty} P_j(\ell)\right] = 0.$$

By the definition of $P_j(i)$, for every $i \le k_0$, $\sum_{\ell=i}^{\infty} P_j(\ell) = \sum_{\ell=i}^{\infty} Q_j(\ell)$. So,

$$\sum_{i=1}^{k_0} \left[\sum_{\ell=i}^{\infty} G_j(\ell) - \sum_{\ell=i}^{\infty} Q_j(\ell)\right] + G_j(k_0+1) - P_j(k_0+1) = 0.$$

The first constraint (Equation (1)) states that $\sum_{\ell=0}^{i} G_j(\ell) \ge \sum_{\ell=0}^{i} Q_j(\ell)$, thus, $\sum_{\ell=i}^{\infty} G_j(\ell) \le \sum_{\ell=i}^{\infty} Q_j(\ell)$. Also, we know that $G_j(k_0+1) \le P_j(k_0+1)$. Since $G_j(i_0) \ne Q_j(i_0)$, we get that the total sum cannot be zero, that is, at least one element in the sum is negative (but none is positive). Therefore, $i_1 > k_0$.

The last case to consider is when $i_0 = k_0$. If $G_j(i_0) < P_j(i_0)$, then $G_j(i)$ clearly does not satisfy the second constraint (Equation (2)) as for every $i < k_0$, $G_j(i_0) = P_j(i_0)$. Therefore, $G_j(i_0) > P_j(i_0)$. Furthermore, the second constraint implies that there must be some integer $i_1 > k_0 + 1$ such that $G_j(i_1) \ne 0$. Therefore, $i_1 - i_0 \ge 2$.

We are now ready to define another distribution, $G'_j(i)$, which also has minimal cost function:

$$G'_j(i) = \begin{cases} G_j(i) - w & i \in \{i_0, i_1\} \\ G_j(i) + w & i \in \{i_0+1, i_1-1\} \\ G_j(i) & \text{otherwise} \end{cases}$$

where $w = \min\{G_j(i_0) - P_j(i_0), G_j(i_1) - P_j(i_1)\}$. Notice that $G'_j(i)$ is well-defined since $i_1 - i_0 \ge 2$. In addition, $w > 0$ since $G_j(i_0) > P_j(i_0)$ and $G_j(i_1) > P_j(i_1)$. Hence, $G'_j(i)$, which clearly preserves both constraints, has a cost no larger than $G_j(i)$. By continuing this process, we end up with $P_j(i)$ no matter what $G_j(i)$ is, as $i_1 - i_0$ decreases at each step by at least 1. This implies that $P_j(i)$ minimizes the cost function.

Finally, since we are interested in the *limit* balancing cost lower bound $\phi_{\text{LB}}^{\text{BAL}}$, we consider the limit distribution $P_{\text{LB}}(i)$ of the distribution $P_j(i)$ that was found to be optimal for any finite parameters. This is done by using the Poisson approximation for the binomial distribution $Q_j(i)$ of the buffers occupancy before the removal process [13], [19], [20], where we use the same approximation to find the values of k_0, p_0 and e_0. Also, by symmetry we get that $E_0 = \frac{(1-\gamma)\cdot n}{m} = r \cdot (1-\gamma)$. ■

Under the assumptions above, we derive the following examples:

Example 1: Figure 1 shows the lower bound distribution $P_{\text{LB}}(i)$ for $r = \frac{n}{m} = 8$, $\gamma = 0$ and $a \in \{1, 1.1, 1.2\}$. As a increases, i.e. as the hashing scheme is allowed increasingly more accesses to perform a better load-balancing, we can see that the lower-bound requires a balancing that is more and more efficient. Note that when $a = 1$, $k_0 = \infty$ because $\gamma = 0$; therefore, for that case, $P(i)$ matches the Poisson distribution with parameter $\lambda = r = 8$, as shown using the solid line.

B. Multiple Hash Function Distributions

We consider a setting where $\ell \le d$ different distributions over the buckets are used by the d hash functions. Denote

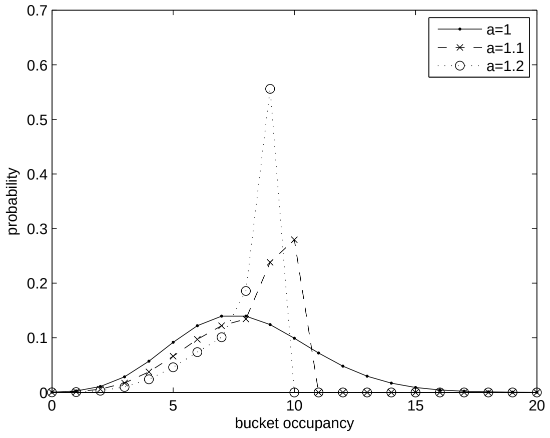

Fig. 1. the lower bound distribution $P_{\text{LB}}(i)$ with $r = 8$ for different values of a

these distributions by $f^1, \ldots, f^\ell$, and assume that distribution f^i is used by a fraction k_i of the total memory accesses, with $\sum_{i=1}^{\ell} k_i = 1$. We now show that Theorem 1 holds also in this case when $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$.

Theorem 2: If $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$ then the optimal expected limit balancing cost $\phi_{\text{OPT}}^{\text{BAL}}$ in the OPTIMAL BALANCING PROBLEM is lower bounded by

$$\phi_{\text{LB}}^{\text{BAL}} = \sum_{j=0}^{k_0+1} P_{\text{LB}}(i) \cdot \phi_B(i).$$

where $P_{\text{LB}}(i)$ is as given in Theorem 1.

Proof: As in the proof of Theorem 1, the number of elements mapped by the hash functions with distribution f_p to bucket i follows approximately a Poisson distribution with rate $k_p \cdot a \cdot n \cdot f_p(i)$. Since the sum of Poisson random variables is also a Poisson random variable, the total number of elements mapped to the bucket i follows a Poisson distribution with rate $an \sum_{p=1}^{l} k_p f_p(i)$.

Thus, the proof of Theorem 1 implies that if for every bucket i, $an \sum_{p=1}^{\ell} k_p f_p(i) = \frac{an}{m}$, we get the same limit lower bound balancing cost $\phi_{\text{LB}}^{\text{BAL}}$.

■

IV. SIMPLE - A SINGLE-CHOICE HASHING SCHEME

We now want to find simple hashing schemes that can potentially achieve the balancing cost lower bound $\phi_{\text{LB}}^{\text{BAL}}$, and therefore the optimal balancing cost $\phi_{\text{OPT}}^{\text{BAL}}$.

We start by analyzing a simplistic hashing scheme, denoted SIMPLE, that is associated with 2 parameters h and p. This scheme only uses a single uniformly-distributed hash function H. Each element is stored in bucket $H(x)$ if it has less than h elements. In case there are exactly h elements, the element is stored in the bucket with probability of p and in the overflow list with probability of $1 - p$. Otherwise, the element is stored in the overflow list.

A. Description by Differential Equations

In recent years, several hashing schemes have been modeled using a deterministic system of differential equations [11], [13], [21]. We adapt this approach and first describe it shortly.

We consider the elements insertion process as performed between the time $t = 0$ and $t = 1$, that is, at time $t = \frac{j}{n}$ the j-th element is inserted. Furthermore, let $F_i\left(\frac{j}{n}\right)$ denote the fraction of buckets in the hash table that store exactly i elements at time $\frac{j}{n}$, just before element j is inserted, and $\vec{F}\left(\frac{j}{n}\right)$ be the vector of all $F_i\left(\frac{j}{n}\right)$'s. Also, let $\Delta F_i\left(\frac{j+1}{n}\right) \stackrel{\triangle}{=} F_i\left(\frac{j+1}{n}\right) - F_i\left(\frac{j}{n}\right)$ denote the change in the fraction of buckets that store exactly i elements between times $\frac{j}{n}$ and $\frac{j+1}{n}$. Then

$$\mathbf{E}\left(\Delta F_i\left(\frac{j+1}{n}\right) | \vec{F}\left(\frac{j}{n}\right)\right) = \begin{cases} -\frac{1}{m} F_0\left(\frac{j}{n}\right) & i = 0 \\ \frac{1}{m}\left(F_{h-1}\left(\frac{j}{n}\right) - p \cdot F_h\left(\frac{j}{n}\right)\right) & i = h \\ \frac{1}{m} \cdot p \cdot F_{h-1}\left(\frac{j}{n}\right) & i = h+1 \\ \frac{1}{m}\left(F_{i-1}\left(\frac{j}{n}\right) - F_i\left(\frac{j}{n}\right)\right) & \text{otherwise} \end{cases} \quad (3)$$

At time $t = 0$, $F_i(0) = 1$ if $i = 0$ and 0 otherwise.

The probability that element j hits a bucket storing i elements is $F_i\left(\frac{j}{n}\right)$. Thus, in the first equation, the fraction of empty buckets decreases when element j reaches an empty bucket, which occurs with probability of $F_0\left(\frac{j}{n}\right)$. Likewise, in the second equality, the fraction of buckets that store h elements increases when element j hits a bucket storing $h-1$ elements (with probability of $F_{h-1}\left(\frac{j}{n}\right)$), and decreases with probability of p when the element hits a buckets storing h elements (with total probability of $p \cdot F_h\left(\frac{j}{n}\right)$). In the third equality, the fraction of buckets storing $h+1$ elements increases with probability of q if element j hits a bucket storing h elements. Last, in all other cases, the fraction of buckets storing i elements increases if element j hits a bucket storing $i-1$ elements, and decreases if it hits a bucket storing i elements. Any such increment or decrement is by a value of $\frac{1}{m}$, thus, all equations are multiplied by $\frac{1}{m}$.

By dividing both sides of the equation by $\frac{1}{n}$ and considering the fact that n is large, so that the values of $\Delta F_i\left(\frac{j+1}{n}\right)$ are comparatively very small, we can use the *fluid limit* approximation, which is often very accurate [21]:

$$\frac{df_i(t)}{dt} = \begin{cases} -\frac{n}{m} f_0(t) & i = 0 \\ \frac{n}{m}\left(f_{i-1}(t) - p \cdot f_i(t)\right) & i = h \\ p \cdot \frac{n}{m} f_{h-1}(t) & i = h+1 \\ \frac{n}{m}\left(f_{i-1}(t) - f_i(t)\right) & \text{otherwise} \end{cases}$$

More formally, let $\vec{f}(t) \stackrel{\triangle}{=} (f_1(t), \ldots, f_d(t))$ be the solution of the above set of linear differential equations when assuming $f_0(0) = 1$ and $f_i(0) = 0$ for each $i \neq 0$. Then, by Kurtz theorems [22]–[24], the probability that $\vec{f}$ deviates from $\vec{F}$ by more than some constant ε decays exponentially as a function of n and ε^2 [21]. For further intuition behind this statement, refer to [21] and [25, Chapter 3.4].

B. *Optimality of the* SIMPLE *Scheme*

Theorem 3: Consider the SIMPLE hashing scheme with m buckets and n elements, and use the notations of k_0, p_0, e_0 and P from Theorem 1. Then for any value of γ, the SIMPLE scheme solves the OPTIMAL BALANCING PROBLEM for $a = 1$ whenever it satisfies the two following conditions:
(i) $h = k_0$;
(ii) p is given by the solution of the following equation:

$$P(k_0) = \frac{e^{-p\cdot r}}{(1-p)^h} - \frac{e^{-r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(r\cdot(1-p))^i}{i!}.$$

Proof: We solve the differential equations one by one, substituting the result of the equation for $\frac{df_i(t)}{dt}$ into the equation for $\frac{df_{i+1}(t)}{dt}$. The first equation depends only on $f_0(t)$, and we get immediately that $f_0(t) = e^{-\frac{n}{m}t}$, or $f_0 = e^{-r\cdot t}$. Each equation for $\frac{df_i(t)}{dt}$, where $i \le h$, depends only on $f_{i-1}(t)$ and $f_i(t)$, and we get that for $i < h$, $f_i(t) = \frac{1}{i!}(r\cdot t)^i e^{-r\cdot t}$.

For $f_h(t)$, we get that for $0 \le p < 1$

$$f_h(t) = \frac{e^{-p\cdot r\cdot t}}{(1-p)^h} - \frac{e^{-r\cdot t}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(r\cdot t\cdot(1-p))^i}{i!}$$

and for $p = 1$,

$$f_h(t) = \frac{1}{h!}(r\cdot t)^h e^{-r\cdot t}.$$

We also use the fact that $\sum_{i=0}^{h+1} f_i = 1$ to get $f_{h+1}(t)$.

By substituting $t = 1$ in $f_i(t)$, for $i < h$, we find that $f_i(1) = \frac{1}{i!}(r)^i e^{-r}$. We note that it is also the probability that an arbitrary bucket stores i elements, and that it is equal to $P_{\text{LB}}(i)$, thus mimicking the distribution of $P_{\text{LB}}(i)$ for $i < h$.

We are left to show that there exists such a $p \in [0,1]$ so that using its value for $f_h(1)$ will result in the exact expression for $P_{\text{LB}}(h)$. When substituting $p = 0$, we get that $f_h(1) = 1 - e^{-r}\sum_{i=0}^{h-1}\frac{r^i}{i!}$ which is clearly larger than $P_{\text{LB}}(h)$ (it is equal when $P_{\text{LB}}(h+1) = 0$). On the other hand, when substituting $p = 1$, we get that $f_h(1) = \frac{1}{h!}(r)^h e^{-r}$ which is lower than $P_{\text{LB}}(h)$. Thus, using the Intermediate Value Theorem (all functions are clearly continuous), there exists some $p \in [0,1]$ such that $f_h(1) = P_{\text{LB}}(h)$. Since $\sum_{i=0}^{h+1} f_i(1) = \sum_{i=0}^{h+1} P_{\text{LB}}(i) = 1$, we get also that $f_{h+1}(1) = P_{\text{LB}}(h+1)$.

■

We finish with the following simple example:

Example 2: When $\gamma = 0$, $k_0 = \infty$, therefore the SIMPLE scheme performs with infinite buckets, and there is no meaning for p.

We next verify the accuracy of our model by simulations. Figure 2 shows the evolution over time of $f_0, \ldots, f_3$ where $r = 2.5$, $p = 0.5$ and $h = 2$, comparing the model with simulated values. In the simulation we used $n = 25,000$, and so $m = 10,000$.

V. GREEDY - A MULTIPLE-CHOICE HASHING SCHEME

We now introduce the GREEDY scheme which is also associated with two parameters h and p. In the GREEDY scheme, we use an ordered set of d hash functions $\mathcal{H} = \{H_1, \ldots, H_d\}$, such that all functions are independent and uniformly distributed. Upon inserting an element x, the scheme successively reads the buckets $H_1(x), H_2(x), \ldots, H_d(x)$, and places x in the first bucket that satisfies one of the following two conditions: *(i)* the bucket stores less than h elements, or, *(ii)* the bucket stores exactly h elements, and x is inserted with probability p. If the insertion algorithm fails to store the element in all the d buckets, x is stored in the overflow list. Last, to keep an average number of memory accesses per element of at most a, the process stops when a total of $a\cdot n$ memory accesses has been reached; the remaining elements are placed in the overflow list.

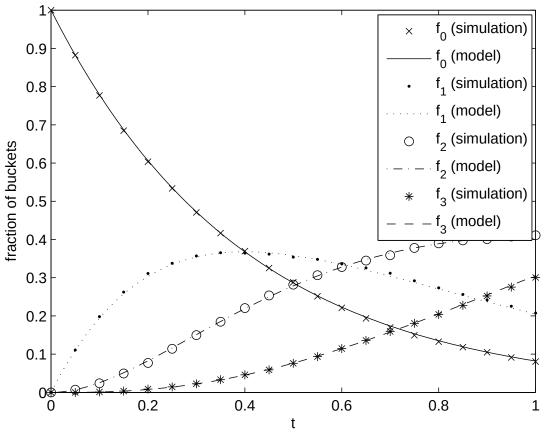

Fig. 2. simulation vs. analytic model for SIMPLE with $r = 2.5$, $p = 0.5$ and $h = 2$

A. *Description by Differential Equations*

We start analyzing the GREEDY scheme by first assuming that there is no constraint on the total number of memory accesses (that is, $a = \infty$) and characterizing the dynamics of the scheme as a system of differential equations.

As before, let $f_i(t)$ represent the fraction of buckets storing i elements at time t, then

$$\frac{df_i(t)}{dt} = \begin{cases} -\frac{n}{m}\cdot f_0\cdot(t)\,g(t) & i = 0 \\ \frac{n}{m}\cdot(f_{h-1}(t) - p\cdot f_h(t))\,g(t) & i = h \\ \frac{n}{m}\cdot p\cdot f_h(t)\,g(t) & i = h+1 \\ \frac{n}{m}\cdot(f_{i-1}(t) - f_i(t))\,g(t) & \text{otherwise} \end{cases} \tag{4}$$

where

$$\begin{aligned} g(t) &= \sum_{k=0}^{d-1}((1-p)\cdot f_h(t) + f_{h+1}(t))^k \\ &= \frac{1-((1-p)\cdot f_h(t) + f_{h+1}(t))^d}{1-((1-p)\cdot f_h(t) + f_{h+1}(t))}, \end{aligned}$$

with $f_0(0) = 1$ and $f_i(0) = 0$ for each $i \neq 0$ as an initial condition. Comparing with the differential equations of the SIMPLE scheme (Equation (3)), there is an additional factor of $g(t)$. For instance, in the first equation, $f_0(t)$ is replaced by

$f_0(t)\,g(t) = \sum_{k=0}^{d-1}\left[\left((1-p)\cdot f_h(t) + f_{h+1}(t)\right)^k \cdot f_0(t)\right]$, which represents the sum of the probabilities of mapping to an empty bucket after being mapped $k = 0, 1, \ldots, d-1$ times to a bucket in which the element could not be stored; in other words, each bucket either already had $h+1$ elements, or had h element but with probability of $1-p$ the element was denied insertion.

We are also interested in the average number of memory accesses performed during this process. Let $f^a_{\text{GREEDY}}(t)$ denote the cumulative number of memory accesses performed by time t, normalized by n. It can be modeled as

$$\frac{df^a_{\text{GREEDY}}(t)}{dt} = \sum_{k=1}^{d-1} k\cdot\left(f_n(t)\right)^{k-1}\left(1-f_n(t)\right) + d\cdot\left(f_n(t)\right)^{d-1}, \quad (5)$$

where

$$f_n(t) = \left((1-p)\cdot f_h(t) + f_{h+1}(t)\right);$$

and $f^a_{\text{GREEDY}}(0) = 0$ as an initial condition.

The differential equation reflects the fact that at a given time t, the cumulative number of memory accesses increases by $1 \le k < d$ memory accesses whenever in the first $k-1$ memory accesses an element is not stored and in the next one the element is stored. It also increases by d memory accesses whenever in the first $d-1$ memory accesses an element is not stored, independently of the bucket state in the d-th memory access.

B. Optimality of the GREEDY Scheme

We now want to show the optimality of the GREEDY scheme over a range of values of a and γ. In general, the above differential equations are hard to solve analytically, and thus cannot help in showing optimality — even though they can of course be solved numerically and yield a numerical approximation of the expected balancing cost.

Instead, to show the optimality of the GREEDY scheme, we reduce it to the optimality of the SIMPLE scheme. Since both the SIMPLE and GREEDY schemes use the same uniform distribution, a new attempt to insert an element after an unsuccessful previous attempt in the GREEDY scheme is equivalent to creating a new element in the SIMPLE scheme and then trying to insert it. In other words, the number of elements successfully inserted by the GREEDY scheme after considering n elements and using a total of $a\cdot n$ memory accesses is the same as the number of elements successfully inserted by the SIMPLE scheme after considering $a\cdot n$ elements.

Theorem 4: Consider the GREEDY hashing scheme with m buckets and n elements, and use the notations of k_0, p_0, e_0 and P from Theorem 1. The GREEDY scheme solves the OPTIMAL BALANCING PROBLEM whenever it satisfies the three following conditions:
(i) $h = k_0$;
(ii) t_0 is such that $f^a_{\text{GREEDY}}(t_0) = a$, and $t_0 \le 1$;
(iii) p is given by the solution of the following equation:

$$P(k_0) = \frac{e^{-p\cdot a\cdot r}}{(1-p)^h} - \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{\left(a\cdot r\cdot(1-p)\right)^i}{i!}.$$

Moreover, the optimality region, that is, the minimum γ needed to achieve optimality of the GREEDY hashing scheme, is given by the overflow list of size $\gamma\cdot n$ that results in $t_0 = 1$, that is, $f^a_{\text{GREEDY}}(1) = a$.

Proof: We compare the GREEDY scheme with the SIMPLE scheme. In the GREEDY scheme we continually try to insert each element, until either it is placed or all d functions are used.

Note that all hash functions have the same (uniform) distribution over all buckets. Thus, for every i, $f_i(t)$ are independent of the exact elements that are hashed. Therefore, applying $d_1 \le d$ hash functions on the same element is equivalent to applying a single hash function on d_1 elements. This implies we can use the results of the SIMPLE scheme, in which a total of $a\cdot n$ elements are considered and therefore a total of $a\cdot n$ memory accesses are performed.

Given an average number of memory accesses a, we set $h = k_0$, and let GREEDY operate until a is reached, that is, until time $t_0 \le 1$ such that $f^a_{\text{GREEDY}}(t_0) = a$. We apply the SIMPLE dynamics to get $f_i(t_0)$, and get that

$$f_i(t_0) = \begin{cases} \frac{1}{i!}(a\cdot r)^i e^{-a\cdot r} & i < h \\ \frac{e^{-p\cdot a\cdot r}}{(1-p)^h} - \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{j=0}^{h-1}\frac{(a\cdot r\cdot(1-p))^j}{j!} & i = h \\ 1 - \sum_{j=0}^{h-1}\frac{1}{j!}(a\cdot r)^j e^{-a\cdot r} - \\ \quad \frac{e^{-p\cdot a\cdot r}}{(1-p)^h} + \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{j=0}^{h-1}\frac{(a\cdot r\cdot(1-p))^j}{j!} & i = h+1 \end{cases} \quad (6)$$

Using the same consideration as in the proof of Theorem 3, we find that there is some $p \in [0,1]$ such that $f_i(t_0)$ is exactly $P_{\text{LB}}(i)$, for every i.

However, note that the GREEDY scheme cannot bring to any desired number of memory accesses a, and is limited to $f^a_{\text{GREEDY}}(1)$. Since $f^a_{\text{GREEDY}}(t)$ increases as k_0 decreases, and k_0 decreases as γ increases, then the minimum γ such that the GREEDY scheme achieves optimality, for a given a and r, is given by γ_0 such that $f^a_{\text{GREEDY}}(1) = a$. Thus, we get the optimality region of this scheme. ■

VI. The Multi-Level Hash Table (MHT) Scheme

The *multi-level hash table (*MHT*)* scheme conceptually consists of d separate subtables $T_1, \ldots, T_d$, where T_i has $\alpha_i \cdot n$ buckets, and d associated hash functions $H_1, \ldots, H_d$, defined such that H_i never returns values of bucket indices outside T_i.

Using the MHT scheme, element x is placed in the smallest i that satisfies one of the following two conditions: *(i)* the bucket $H_i(x)$ stores less than h elements, or, *(ii)* the bucket $H_i(x)$ stores exactly h elements, and element x is then inserted with probability p. If the insertion algorithm fails to store the element in all the d buckets, x is placed in the overflow list. Since that smallest i with available space is used, the memory accesses for each element x are sequential, starting from $H_1(x)$ until a place is found or all d hash functions are used (and the element is stored in the overflow list).

A. Description by Differential Equations

The system of differential equations that characterizes the dynamics of MHT is influenced by the static partitioning of the memory among subtables, which introduces extra variables. Specifically, let $f_{i,j}(t)$ be the fraction of buckets in subtable T_j that store exactly i elements. Then:

$$\frac{df_{i,j}(t)}{dt} = \begin{cases} -\frac{n}{\alpha_j m} f_{0,j}(t)\, g_j(t) & i=0 \\ \frac{n}{\alpha_j m}\left(f_{h-1,j}(t) - p\cdot f_{h,j}(t)\right) g_j(t) & i=h \\ \frac{n}{\alpha_j m}\cdot p\cdot f_{h,j}(t)\, g_j(t) & i=h+1 \\ \frac{n}{\alpha_j m}\left(f_{i-1,j}(t) - f_{i,j}(t)\right) g_j(t) & \text{otherwise} \end{cases} \tag{7}$$

where

$$g_j(t) = \prod_{k=1}^{j-1}\left((1-p)\, f_{h,k}(t) + f_{h+1,k}(t)\right)$$

represents the probability that all the insertion attempts in subtables $T_1, \cdots, T_{j-1}$ do not result in storing the element, and thus that MHT will attempt to insert the element in subtable T_j. By convention $g_1(t) = 1$. The initial conditions are $f_{i,j}(0) = 1$ for $i = 0$ and $f_{i,j}(0) = 0$ otherwise.

As in the GREEDY scheme, let $f^a_{\text{MHT}}(t)$ denote the cumulative number of memory accesses done by time t, normalized by n. Then the following differential equation reflects the dynamics of $f^a_{\text{MHT}}(t)$:

$$\frac{df^a_{\text{MHT}}(t)}{dt} = \sum_{k=1}^{d-1} k\cdot g_k(t)\left(1-(1-p)\, f_{h,k}(t) - f_{h+1,k}(t)\right) + d\cdot g_d(t), \tag{8}$$

with $f^a_{\text{MHT}}(0) = 0$.

B. *Reduction to the* SIMPLE *Scheme*

In the last section we presented the description of the dynamics of the MHT scheme using a system of differential equations; however, this system is generally difficult to solve. This can be circumvented by relying on our results from the SIMPLE scheme.

Our approach relies on the fact that *each subtable follows a local* SIMPLE *scheme*. More specifically, all elements attempting to access some subtable T_j only access a single uniformly-distributed bucket in T_j, and if this bucket is full, do not consider any other bucket in T_j. Thus, within each subtable T_j, MHT behaves like SIMPLE, with a number of initial elements that depends on previous subtables.

More formally, let $f_i^{\text{SIMPLE}}(t)$ be the fraction of buckets that store exactly i elements at time t in the SIMPLE scheme. As in the proof of Theorem 3, it is given by:

$$f_i^{\text{SIMPLE}}(t) = \begin{cases} \frac{1}{i!}(r\cdot t)^i e^{-r\cdot t} & i<h \\ \frac{e^{-p\cdot r\cdot t}}{(1-p)^h} - \frac{e^{-r\cdot t}}{(1-p)^h}\sum_{j=0}^{h-1}\frac{(r\cdot t\cdot(1-p))^j}{j!} & i=h \\ 1-\sum_{j=0}^{h-1}\frac{1}{j!}(r\cdot t)^j e^{-r\cdot t} - \frac{e^{-p\cdot r\cdot t}}{(1-p)^h} + \frac{e^{-r\cdot t}}{(1-p)^h}\sum_{j=0}^{h-1}\frac{(r\cdot t\cdot(1-p))^j}{j!} & i=h+1 \end{cases} \tag{9}$$

Also, let $\gamma^t_{\text{SIMPLE}}(t)$ be the fraction of the elements that are not stored in the buckets out of all the elements that arrived up to time t. Given all $f_i^{\text{SIMPLE}}(t)$'s, it is simply the complementary of the expectation of the number of elements in the buckets up to time t normalized by the total number of elements arrived up to this time:

$$\gamma^t_{\text{SIMPLE}}(t) = 1 - \frac{m}{nt}\cdot\sum_{i=0}^{h+1} i f_i^{\text{SIMPLE}}(t). \tag{10}$$

Let $n_j(t)$ denote the number of elements that are considered in subtable T_j up to time t, and $\gamma^t_j(t)$ denote the fraction of these elements that are not placed in subtable T_j. We will express these using f_i^{SIMPLE} and γ^t_{SIMPLE}, the corresponding functions in the SIMPLE scheme.

Note that as shown in Equations (9) and (10), $f_i^{\text{SIMPLE}}(t)$ and $\gamma^t_{\text{SIMPLE}}(t)$ only depend on the time t, the number of elements n, the number of buckets m, the bucket size h and the probability p; thus, we refer to them as $f_i^{\text{SIMPLE}}(t,m,n,h,p)$ and $\gamma^t_{\text{SIMPLE}}(t,m,n,h,p)$. We obtain the following theorem, which is valid for any arbitrary partition of the subtables.

Theorem 5: Consider an $\langle a,d,r\rangle$ MHT hashing scheme in which for each $1\le j\le d$, subtable T_j has $\alpha_j\cdot m$ buckets, with $\sum\alpha_j = 1$. Then, as long as $f^a_{\text{MHT}}(t)\le a$, the functions $n_j(t)$, $\gamma^t_j(t)$ and $f_{i,j}(t)$ satisfy

$$n_j(t) = n\cdot t\cdot\prod_{k=1}^{j-1}\gamma^t_k(t), \tag{11}$$
$$\gamma^t_j(t) = \gamma^t_{\text{SIMPLE}}(1,\alpha_j m, n_j(t), h, p), \tag{12}$$
$$f_{i,j}(t) = f_i^{\text{SIMPLE}}(1,\alpha_j m, n_j(t), h, p). \tag{13}$$

Proof: By the definition of the MHT scheme, it follows immediately that $n_j(t) = \gamma^t_{j-1}(t)\, n_{j-1}(t)$; since $n_1(t) = n\cdot t$ (all elements go through the first subtable), we get that $n_j(t) = n\cdot t\cdot\prod_{k=1}^{j-1}\gamma^t_k(t)$.

Equations (12) and (13) are immediately derived by setting the right parameters for each SIMPLE scheme within each subtable T_j; namely, its total number of buckets is $\alpha_j\cdot m$ and the number of elements by time t is $n_j(t)$. ■

C. *Optimality of the* MHT *Scheme*

We now show that MHT is optimal on a given range of a and γ.

Theorem 6: Consider an $\langle a,d,r\rangle$ MHT hashing scheme in which each subtable T_j has $\alpha_j\cdot m$ buckets, with $\sum\alpha_j = 1$, and use the notations of k_0, p_0, e_0 and P from Theorem 1. Further, let $p(a) = \gamma^t_{\text{SIMPLE}}(1,m,a\cdot n,h,p)$ denote the overflow fraction of the SIMPLE scheme with $a\cdot n$ elements. Then, the $\langle a,d,r\rangle$ MHT scheme solves the OPTIMAL BALANCING PROBLEM whenever it satisfies the following four conditions:
(i) $h = k_0$;
(ii) $t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right)$, and $t_0\le 1$;
(iii) p is given by the solution of the following equation:

$$P(k_0) = \frac{e^{-p\cdot a\cdot r}}{(1-p)^h} - \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(a\cdot r\cdot(1-p))^i}{i!};$$

(iv) the subtable sizes $\alpha_j \cdot m$ follow a geometric decrease of factor $p(a)$:

$$\alpha_j = \left(\frac{1 - p(a)}{1 - p(a)^d} \right) p(a)^{j-1}. \tag{14}$$

Moreover, the optimality region (the minimum γ needed to achieve optimality of the MHT hashing scheme) is given by the overflow list of size $\gamma \cdot n$ that results in $t_0 = 1$, that is, $f^a_{\text{MHT}}(1) = a$. Furthermore, if all four conditions are met then all buckets have an identical occupancy distribution.

Proof: Given the average number of memory accesses a, we set $h = k_0$. We would like to let MHT operate until a is reached, that is, until time $t_0 \leq 1$ such that $f^a_{\text{MHT}}(t_0) = a$. However, $f^a_{\text{MHT}}(t)$ depends on the subtables sizes α_j's.

Up to time t_0, in which we aim to exhaust all $a \cdot n$ memory accesses, we used exactly $n_j(t_0)$ times the hash function $H_j(x)$ in subtable T_j. Since we aim at an optimal balancing cost, the necessary condition on the distributions of the hash functions, given in Theorem 2, immediately implies that

$$\alpha_j = \frac{n_j(t_0)}{n \cdot a}. \tag{15}$$

By substituting the expression for α_j in (11), we get:

$$\begin{aligned}
\gamma^t_j(t_0) &= \gamma^t_{\text{SIMPLE}}\left(1, \frac{n_j(t_0)}{n \cdot a} m, n_j(t), h, p\right) \\
&= 1 - \frac{\frac{n_j(t_0)}{n \cdot a} m}{n_j(t)} \cdot \\
&\quad \sum_{i=0}^{h+1} i f^{\text{SIMPLE}}_i \left(1, \frac{n_j(t_0)}{n \cdot a} m, n_j(t), h, p\right) \\
&= 1 - \frac{m}{n \cdot a} \cdot \sum_{i=0}^{h+1} i f^{\text{SIMPLE}}_i (1, m, a \cdot n, h, p) \\
&= \gamma^t_{\text{SIMPLE}}(1, m, a \cdot n, h, p) \\
&= p(a)
\end{aligned}$$

It is important to notice that, quite surprisingly, $\gamma^t_j(t_0)$ does not depend on j.

We now obtain the time t_0 by observing that $n_j(t_0) = n \cdot t_0 \cdot p(a)^{j-1}$, thus $\alpha_j = \frac{t_0 \cdot p(a)^{j-1}}{a}$. Since $\sum_{k=1}^d \alpha_k = 1$, we get $\sum_{k=1}^d \frac{t_0 p^{j-1}}{a} = 1$, and therefore t_0 is given by the sum of a geometric series:

$$t_0 = a \left(\frac{1 - p(a)}{1 - p(a)^d} \right). \tag{16}$$

This, in turn, immediately gives us the claimed memory partitioning α_j.

We now turn to show that all bucket distributions are identical. In subtable T_j, the total number of elements considered is $n_j(t_0) = n \cdot t_0 \cdot p(a)^{j-1}$, while there are $m \cdot \alpha_j = m \cdot \left(\frac{1-p(a)}{1-p(a)^d}\right) p(a)^{j-1}$ buckets. Hence, by Theorem 5, we get that the fraction of buckets in subtable T_j that store exactly i elements $f_{i,j}(t)$ is given by

$$f^{\text{SIMPLE}}_i \left(1, m \cdot \left(\frac{1 - p(a)}{1 - p(a)^d}\right) p(a)^{j-1}, n \cdot t_0 \cdot p(a)^{j-1}, h, p\right)$$

and by substituting $t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right)$, we get that $f_{i,j}(t) = f^{\text{SIMPLE}}_i(1, m, a \cdot n, h, p)$.

Finally, as in the proof of Theorem 4, the MHT scheme cannot bring to any desired average number of memory accesses a, but is limited to $f^a_{\text{MHT}}(1)$. Since $f^a_{\text{MHT}}(t)$ increases as k_0 decreases, and k_0 decreases as γ increases, then the minimum γ such that the MHT scheme achieves optimality, for a given a and r, is given by γ_0 such that $f^a_{\text{MHT}}(1) = a$. Since $t_0 = a\left(\frac{1-p(a)}{1-p(a)^d}\right)$, we seek γ such that $a = \frac{1-p(a)^d}{1-p(a)}$. Thus, we get the optimality region of this scheme. ∎

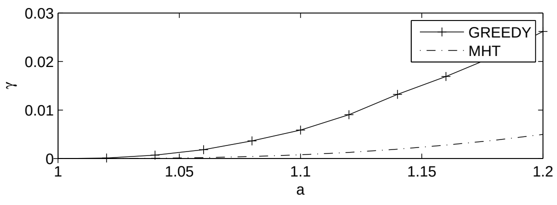

Fig. 3. Optimality regions of GREEDY and MHT with $r = 8$ and $d = 3$

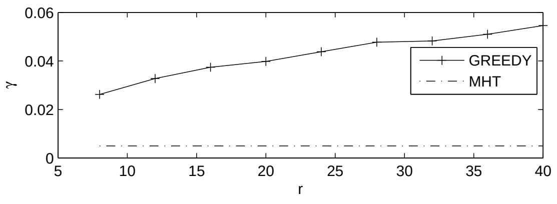

Fig. 4. Optimality regions of GREEDY and MHT with $a = 1.2$ and $d = 3$

VII. Comparative Evaluation

Figure 3 shows the optimality region of GREEDY and MHT with $r = 8$ and $d = 3$, that is, for each value of the average number of memory accesses a, it shows the minimum value of γ that suffices to solve the OPTIMAL BALANCING PROBLEM. We can see that for a CAM size of 1%, GREEDY achieves optimality for $a \approx 1.1$. We will show in Section IX that the optimal solution for such parameters dramatically reduces the balancing cost compared to $a = 1$ with no CAM.

On the other hand, Figure 4 shows the optimality region of GREEDY and MHT with $a = 1.2$ and $d = 3$ for different values of r. We can see that MHT scales better to higher loads.

VIII. Variance of the Query-Time in Chain-Based Hash-Tables

The balancing problem can be directly used in order to construct an energy-constrained hashing scheme with optimal variance over its query time. In such a scheme, the time it takes to complete a *lookup* operation directly depends on the occupancy of the buckets. For example, suppose that there is only one hash-function, and some element y is mapped to a bucket $h(y)$; in order to query y we need to go over all elements of $h(y)$ in case y is not in the table, or on average over half of them in case y is in the hash-table.

Note that the *average load* is simple to obtain and equals $\frac{(1-\gamma)n}{m}$. We next show how to find hashing schemes with minimal variance Var(O). This is simply done by defining the appropriate block cost function:

$$\phi_B(i) = i^2 - \left(\frac{(1-\gamma)\,n}{m}\right)^2 .$$

Thus,

$$\begin{aligned} \phi^{\text{BAL}} &= \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} \mathbf{E}\left(O_j^2 - \left(\frac{(1-\gamma)\,n}{m}\right)^2\right) \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} \mathbf{E}(O_j^2) - \mathbf{E}(O_j)^2 \\ &= \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} \text{Var}(O_j) \\ &= \text{Var}(O), \end{aligned}$$

where O_j is the random variable representing the occupancy of bucket j. By symmetry, all variables O_j have the same distribution and thus the same variance, which is denoted by Var(O).

This immediately implies that the schemes we presented can be used in order to build a hash-table with optimal variance.

IX. Energy-Efficient Bloom Filters

A. Architecture

The standard Bloom filter is not energy efficient when wide memory words can be read and written. If such a memory is used, there are potentially k memory words that have to be read for an insertion (and query operation), since it is most likely that the k hash functions point to distinct memory words (when the filter is sufficiently large). To avoid this problem, a *blocked Bloom filter* [14] first picks a single memory word where all hash functions point to the bits within this memory word, thus, only one memory word is read (or written).

The blocked Bloom filter mechanism corresponds to the SIMPLE scheme with $\gamma = 0$ and block cost function $\phi_B(i) = \left(1 - \left(1 - \frac{1}{B}\right)^{ki}\right)^B \approx \left(1 - e^{-\frac{ki}{B}}\right)^k$, where B is the size of the memory word (in bits) and k is the number of hash functions used. However, although the SIMPLE scheme is optimal, its average number of memory accesses is 1, thus, it achieves poor balancing of the elements that results in a poor false positive rate. In this section we suggest to use the optimal online hashing schemes to achieve a better balancing between the buckets, and therefore, better false positive rate.

Figure 5 illustrates an insertion of a new element into a Bloom filter based on the MHT hashing scheme with $h = 2$ and some arbitrary p. The memory consists of 3 subtables of decreasing size, the first one with 4 memory words, the second with 2 memory words and the last one with one memory word. Each memory word is of size 6 bits, where 2 bits are used as a counter that counts the number of elements already inserted into the memory word. When element x arrives, it is first hashed to the memory word at address 10 in subtable T_1. The counter at this memory word indicates that 3 elements have already been inserted there ('11' in binary basis), therefore the schemes tries to insert the element into subtable T_2. In this

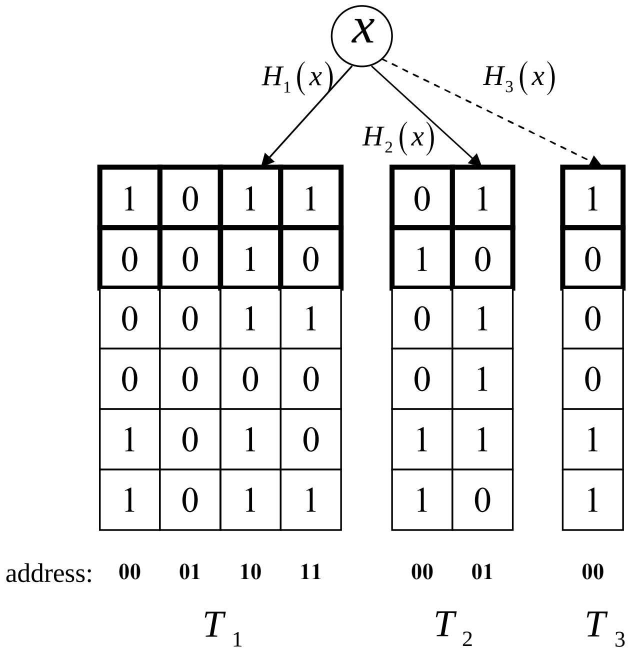

Fig. 5. Illustration of a Bloom filter implementation using MHT.

subtable, the element is hashed to address 01, where there are 2 elements (the counter indicates '10' in binary basis). At this point, the element is inserted to this memory word in probability of p, which is indeed the case in this illustration. The dashed arrow to subtable T_3 illustrates a hash function that is not actually performed.

B. Implementation Considerations

Assuming memory words of size B bits and bits-per-element ratio β, the number of elements per bucket r is given by B/β. Using the optimal online hashing schemes to implement a Bloom filter requires saving some $b = \lceil \log_2 (k_0 + 2) \rceil$ bits in every memory word to count the elements hashed into each one. Thus, the block cost function is given by $\phi_B(i) = \left(1 - e^{-\frac{ki}{B-b}}\right)^k$.

In the standard Bloom filter, the optimal false positive rate is archived when using $k = r \cdot \ln 2$ hash functions [17]. Although this may not be the best choice when using large memory words as one has to take into consideration the distribution of the number of elements in each bucket, we will use the same k for simplicity.

Hashing schemes that read more than one memory word on a query operation increase the false positive rate, because a false positive may result from a query operation on each one of the buckets. Thus, since the probability of false positive in every memory word is relatively very small, then the overall false positive probability, i.e. the false positive rate, is given by $\sum_{j=1}^{d} P_j \cdot \text{FPR}_j$, where P_j is the probability that a query operation results in at least j memory words, and FPR_j is the false positive probability of the j-th memory word read.

C. Comparative Analysis

Figure 6 compares the false positive rate for different values of bits-per-element ratios with memory word size $B = 256$.

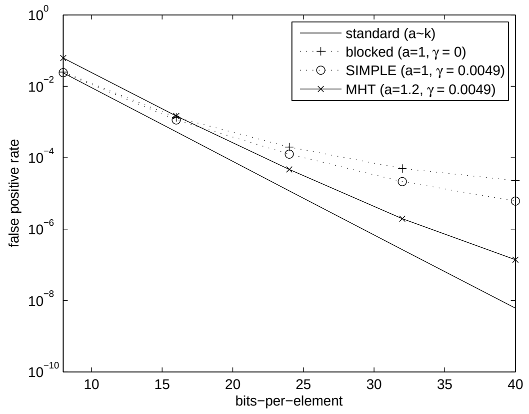

Fig. 6. False positive rate of different Bloom filter schemes with memory word size of 256 bits. SIMPLE and MHT perform with $\gamma = 0.49\%$.

The MHT hashing scheme is used with $a = 1.2$ and $d = 3$. All these parameters let us compute the minimum overflow list size $\gamma \cdot n$ such that the scheme is optimal. As Figure 4 shows, γ is independent of the elements-per-bucket ratio r, and therefore, independent of the bits-per-element ratio, so the overflow list size used is $\gamma = 0.49\%$. For comparison, the SIMPLE hashing scheme is used with $a = 1$ and the same overflow list size, that is, $\gamma = 0.49\%$.

As Figure 6 indicates, the SIMPLE hashing scheme performs slightly better than the blocked Bloom filter scheme. However, the MHT scheme performs worse for low values of bits-per-element ratio. This is due to the need to check multiple memory words (up to d) on a query operation. But for larger values of bits-per-element ratios, the MHT performs better by two orders of magnitude, and only one order of magnitude worse then the standard Bloom filter, which uses a memory-inefficient $a \approx k$.

For example, for a bit-per-element ratio of 24, $k = 17$ hash functions are used, introducing up to 17 memory-read operations in the standard Bloom filter, which can clearly present memory-throughput and power-consumption issues. This problem worsens for a bit-per-element ratio of 40, where $k = 28$ hash functions are used. In this case, the false positive rate of the standard Bloom filter is $6.1 \cdot 10^{-9}$ while the corresponding false positive rate of MHT is $1.4 \cdot 10^{-7}$ and of the blocked Bloom filter is $2.3 \cdot 10^{-5}$.

X. Conclusion

In this paper we presented the balancing problem, which deals simultaneously with several system parameters: the worst-case insertion time, the energy budget, the overflow list size, and the targeted utility functions. Furthermore, depending on the specific given parameters, we showed how to solve this problem optimally, by applying a simple approach (using only a single hash function), then a greedy approach, and last by building the hash table in hierarchical manner as in the *mutli-level hash table*.

Since the cost function is very general, the balancing problem can be used in many contexts. We first showed how it can be used in order to find optimal hashing schemes that take into the account the variance of the query-time and not just its expected or worst-case time. Then, we showed that when setting the cost function to correspond to the *false positive rate* of a Bloom Filter, the balancing problem can be used to devise energy-efficient Bloom Filter architectures.

A promising line of future research is to find other applications for the balancing problem. For example, we believe that the balancing problem can prove very effective for distributed storage systems and datacenters, where load balancing plays a major role.

References

[1] Intel Corp., "Increasing data center density while driving down power and cooling costs," 2006.
[2] N. L. Johnson and S. Kotz, *Urn models and their application: an approach to modern discrete probability theory*. Wiley New York, 1977.
[3] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," in *ACM STOC*, 1994, pp. 593–602.
[4] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More robust hashing: Cuckoo hashing with a stash," in *ESA*, 2008, pp. 611–622.
[5] Y. Azar, "On-line load balancing," *Theoretical Computer Science*, pp. 218–225, 1992.
[6] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, "On-line load balancing with applications to machine scheduling and virtual circuit routing," in *ACM STOC*. New York, NY, USA: ACM, 1993, pp. 623–631.
[7] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, p. 1563.
[8] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, "Load balancing in dynamic structured p2p systems," in *INFOCOM*, 2004.
[9] I. Keslassy, "The load-balanced router," Ph.D. dissertation, Stanford, CA, USA, 2004.
[10] G. H. Gonnet, "Expected length of the longest probe sequence in hash code searching," *J. ACM*, vol. 28, no. 2, pp. 289–304, 1981.
[11] M. Mitzenmacher, A. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*, 2000, pp. 255–312.
[12] A. Kirsch, M. Mitzenmacher, and G. Varghese, *Hash-Based Techniques for High-Speed Packet Processing*. DIMACS, 2009, to appear. [Online]. Available: http://www.eecs.harvard.edu/ michaelm/postscripts/dimacs-chapter-08.pdf
[13] Y. Kanizo, D. Hay, and I. Keslassy, "Optimal fast hashing," in *IEEE Infocom*, 2009.
[14] F. Putze, P. Sanders, and J. Singler, "Cache-,hash- and space-efficient Bloom filters," in *Workshop on Experimental Algorithms*, 2007, pp. 108–121.
[15] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.
[16] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[17] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
[18] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Beyond Bloom filters: from approximate membership checks to approximate state machines," in *SIGCOMM*, 2006, pp. 315–326.
[19] A. D. Barbour and P. G. Hall, "On the rate of Poisson convergence," *Math. Proc. Cambridge Philos. Soc.*, vol. 95, no. 3, pp. 473–480, 1984.
[20] J. M. Steele, "Le Cam's inequality and Poisson approximations," *American Mathematical Monthly*, vol. 101, pp. 48–54, 1994.
[21] A. Kirsch and M. Mitzenmacher, "The power of one move: Hashing schemes for hardware," in *IEEE Infocom*, 2008, pp. 565–573.
[22] S. N. Ethier and T. G. Kurtz, *Markov processes*. John Wiley&Sons, 1986.
[23] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *J. of Applied Probability*, vol. 7, no. 1, pp. 49–58, 1970.
[24] ——, *Approximation of Population Processes*, 1981.
[25] M. Mitzenmacher, "The power of two choices in randomized load balancing," Ph.D. dissertation, University of California at Berkley, 1996.