Consistent Load Balancing Via Spread Minimization

Dedicated to Danny Lewin.

Robert Kleinberg [*] Tom Leighton [†]

ABSTRACT

Motivated by applications to web caching and other distributed server architectures, we analyze load balancing algorithms from the standpoint of *spread*, which measures the number of different assignments an item receives across multiple iterations of the algorithm on varying load distributions. Minimizing spread while balancing load is important in order to make efficient use of server resources such as memory and in order to minimize service latency. In the paper, we consider both on-line and off-line versions of the problem. Most importantly, we describe on-line load balancing algorithms with very low spread, which means that the assignments made for most items do not change even when the loads associated with the items do change. This means that load balancing is an example of a problem for which it is possible to find highly stable (or, consistent) on-line algorithms — i.e. algorithms for which the output changes only slightly (with high probability) even if the inputs change dramatically.

This paper is dedicated to the memory of Danny Lewin, who pioneered the notion of consistent hashing and its applications to load balancing and content distribution on the Internet. Danny's ideas furnished many of the underpinnings for the problem and algorithms studied herein.

[*]MIT Department of Mathematics, 77 Massachusetts Ave., Cambridge, MA 02139. email: rdk@math.mit.edu. Supported by a Fannie and John Hertz Foundation Fellowship.

[†]Akamai Technologies, 8 Cambridge Center, Cambridge, MA 02142; and MIT Department of Mathematics, 77 Massachusetts Ave., Cambridge, MA 02139. email: ftl@math.mit.edu.

STOC '03, June 9–11, 2003, San Diego, California, USA.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

General Terms

Algorithms, Theory

Keywords

load balancing, randomized algorithms, on-line algorithms

1. INTRODUCTION

A central design issue in distributed web services, such as caching and edge computing, is the need for highly-stable on-line algorithms whose outputs change by a limited amount as the inputs to the algorithm change. This theme appeared in the work of Lewin [6] and of Karger et al [5] on consistent hashing, i.e. hash functions which don't change their output too much in response to changes in the set of hash targets. We believe it is interesting to explore the theme of consistent optimization algorithms more generally. In this paper, we extend the class of consistent algorithms to include load-balancing problems which arise naturally in distributed web applications.

1.1 The model

We consider a load balancing problem in which n items are to be fractionally assigned to m buckets with capacities $\rho_1, \ldots, \rho_m$, in each of t consecutive assignment rounds. The weights of the items vary from round to round, with $\{\mu_{is}\}_{i=1}^{n}$ representing the weights in round s. Letting x_{ijs} denote the fraction of item i which is assigned to bucket j in round s, we may define the *unnormalized spread* of an assignment to be the sum, over all items i, of the number of buckets which receive a positive fraction of the load on i in any round, i.e. the cardinality of the set $\{(i,j) : \exists s\ x_{ijs} > 0\}$. It is technically more convenient to work with the *normalized spread* (hereinafter called simply "spread") which is defined by

$$Spread = \#\{(i,j) : \exists s\ x_{ijs} > 0\} - n.$$

In other words, to compute the normalized spread of an assignment, each item is allowed to use one bucket for free and is charged one unit of spread for each additional bucket used in any round.

The definition of spread is motivated by web caching applications, in which each item represents a web object, its weight (or load) represents the amount of demand for that object, and each bucket represents a web server. In such a context, it is necessary to balance the load of the objects (by assigning each item to one or more buckets) so that no server has a demand that exceeds its capacity. (Here, we assume that fractional assignments of items to buckets cause the loads associated with those items to be apportioned among the corresponding servers accordingly.) The spread of the algorithm measures how often the algorithm causes duplicate copies of an object to be stored on different servers. Thus, minimizing spread corresponds to using the available cache space as efficiently as possible. In the cases of interest, n is very large (corresponding to the number of items that might be accessed or cached), m is fairly small (corresponding to the number of servers), and t is moderate (depending on the frequency of updates, and on the time it takes the cached data to expire, and/or the time it takes to provide updated server information to a client).

Spread-minimization is even more critical in designing load-balancing algorithms for other distributed web services, such as edge computing and grid computing applications. This is because the applications in such cases tend to have a large footprint (a single instance of the application consumes a large amount of memory and other resources), and the cost of moving an application to an extra server is high (e.g. because of the high start-up time).

In these applications, it is natural to model the loads and capacities as multi-variable quantities. For instance, bandwidth and CPU could be regarded as separate variables in any of the applications discussed above. We will consider the spread-minimization problem in a setting where the loads and capacities are d-dimensional vectors for some $d > 1$. Surprisingly, the bounds we obtain in the multivariate case are not much worse than those obtained in the single-variable case.

1.2 Lower bounds for spread

It is easy to see that no algorithm can achieve spread less than $m-1$ in the worst case when $d = t = 1$; simply consider the case where each bucket has unit capacity, one of the items has weight greater than $m-1$, and the rest have weight ε for some small ε. It is also easy to see that there is an algorithm whose worst-case spread is exactly $m-1$, namely the greedy algorithm which assigns items to bin 1 until it fills up, then assigns items to bin 2 until it fills up, and so on. To see this, note that if $x_{ij} > 0$ then either j is the last bucket receiving load from i, or i is the last item assigning load to j. There are n pairs meeting the first criterion and m pairs meeting the second criterion, and the two sets overlap in at least one pair (i, j), namely the last bucket to receive load from the last item. This puts an upper bound of $n + m - 1$ on the unnormalized spread, as desired.

Returning to the case of multiple assignment rounds, there is a lower bound of $t(m-1)$ on the worst-case spread (provided $n \geq t$): simply choose a different heavily-loaded item in each round. Although the spread achieved by the greedy algorithm no longer matches this lower bound, we shall see in Section 3 that there is another algorithm, inspired by the Beck-Fiala Theorem [1], which achieves it exactly.

1.3 The on-line case

The crux of this paper, however, concerns the spread-minimization problem in the framework of on-line algorithms. We always assume that such algorithms know the complete load distribution (i.e. the loads $\{\mu_{is}\}_{i=1}^{n}$) for the present iteration and all prior iterations, as well as their decisions in prior iterations, but they do not know the load distribution in future iterations. We will consider two natural variants of the on-line spread minimization problem.

- A deterministic on-line algorithm running against an omniscient adversary (i.e. one with knowledge of the algorithm's decisions on prior iterations).
- A randomized on-line algorithm running against an oblivious adversary with no knowledge of the algorithm's prior decisions.

The latter variant is the one which is most relevant in the applications.

For the first variant, we shall see that the worst-case spread can be no better than linear in n, even when $m = 2$ and $t = 2$. This is not especially surprising since by changing the loads associated with each item, we might expect that an adversary can force the algorithm to change the assignments for a substantial number of the items.

More importantly, against an oblivious adversary, if the total capacity of the buckets is assumed to exceed the total load of the items by a factor of at least $1 + \varepsilon$, there is a randomized algorithm achieving spread $O_\varepsilon(mt)$ with arbitrarily high probability. This is the best possible spread to within a constant factor, since the lower bound sketched above can easily be modified to give spread $tm/(1+\varepsilon)$ in the case where there is a $(1+\varepsilon)$-factor of surplus capacity.

At first glance, the fact that we can achieve spread $O(mt)$, which is indepedent of n, is somewhat surprising. When n is much larger than mt (as it often is in practice), this means that the algorithm assigns almost every item to a single bucket and that this bucket does not change even if the loads associated with the items change dramatically. In other words, the algorithm is highly stable (or consistent) in the sense that the output changes very little even when the input changes a lot. This is somewhat unexpected since, a priori, load balancing seems like a problem for which the output should be fairly sensitive to the input loads for the items. The low spread is quite useful in practice since it means that most items are assigned to a single server and that this server does not change over time, even if the loads associated with the items change substantially.

1.4 Related work

The subject matter of this paper has precursors in several existing lines of work. The most salient of these is consistent hashing ([5], [6]). Our definition of spread agrees closely with the spread of a random ranged hash function as defined in those papers. Indeed, consistent hashing can be viewed as solving the spread-minimization problem for a set of items of equal weight, when the variable which changes from iteration to iteration is the set of buckets to which the items may be assigned. In contrast, the present paper focuses on assignment problems where the items have differing weights which vary from iteration to iteration. However, there are strong parallels between the algorithms considered here and those which have been used for consistent hashing. In particular, the randomized on-line algorithm analyzed in Section 2.1 is inspired by the use of random permutations in the hash functions constructed in [6].

Another precursor is discrepancy theory ([1], [2], [9]) in which one studies a problem for which exact fractional solutions exist, and the aim, broadly speaking, is to examine how close integer solutions can come to satisfying the constraints of the problem. (For example, one has a collection of subsets of some finite set S, and one wants to color the elements of S red and blue so that each of the given subsets receives approximately equal numbers of each color. This example roughly corresponds to the case $m = 2$ in our load balancing problem, with each subset in the given collection corresponding to the set of heavily-loaded items in one of the t assignment rounds.) In the present context, an integer solution corresponds to a zero-spread assignment. Whereas discrepancy theory concerns itself with algorithms and existence theorems for integer solutions which are as close as possible to being exact solutions, our spread-minimiziation algorithms attempt to find exact solutions which are (in a suitable sense) as close as possible to being integer solutions.

The off-line spread minimization algorithm of Section 3 has qualitative connections with the approximation algorithm for scheduling on unrelated machines developed by Lenstra, Shmoys, and Tardos in [7] and generalized by Shmoys and Tardos in [10]. Although their papers do not mention the notion of spread explicitly, it is implicit in the analysis of their rounding technique for the linear programming relaxation.

Spread minimization is also related to bin packing, on which there is an extensive literature. (See [3] for a survey.) Our problem is the same as bin packing, except that we allow fractional assignment of items and we are trying to get an assignment that is consistent, i.e. that changes very little when the item sizes change. The multivariate case of our algorithm may be of particular interest to the bin packing problem. It is also interesting to note that, in contrast to many bin packing algorithms which pack the large items first, we find it beneficial to pack the small items first.

The theme of algorithms which can change their decisions, but which are discouraged from doing so, arises also in the area of scheduling with pre-emption; see the survey paper [4]. Although this work is thematically connected with ours, the technical issues which arise are quite different.

2. ON-LINE ALGORITHMS

The spread-minimization problem is most interesting in the context of a randomized on-line algorithm running against an oblivious adversary, which is also the context most relevant to the applications. Throughout this section, we will assume that the total capacity of the buckets exceeds the total load in each round by a factor of $1 + \varepsilon$, where ε is some fixed positive constant less than $1/2$. Our bounds on spread will explicitly indicate their dependence on ε. (Note that, in the presence of this surplus capacity condition, the naive lower bound on worst-case spread is $t(m-1)/(1+\varepsilon)$, via the same argument sketched in the introduction.)

Before treating the case of an oblivious adversary, we wish first to present an example indicating that an adversary with knowledge of the algorithm's decisions in prior assignment rounds is "too powerful," in the sense that such an adversary can force $O(n)$ spread, even in the case $m = t = 2$.

EXAMPLE 2.1. *Consider the following instance of the on-line spread minimization problem, with $m = t = 2$ and with both buckets having unit capacity. In the first round, all items have weight $2/n$. Color items red, blue, or purple according to whether the algorithm assigns their load to the first bucket, the second bucket, or both buckets in the first round. Let R, B, P denote the number of red, blue, and purple items, respectively, and assume without loss of generality that $R \geq B$.*

In the second round, place load $2/R$ on each of the red items, and zero load on all other items. To avoid overloading buckets, at least $(1-\varepsilon)R/2$ of the red items must assign some of their load to the second bucket. The spread is equal to the number of items which used both buckets over the course of the two rounds. Thus

$$\begin{aligned} \textit{Spread} &\geq (1-\varepsilon)R/2 + P \\ &\geq (1-\varepsilon)R/4 + (1-\varepsilon)B/4 + P \\ &\geq (1-\varepsilon)n/4. \end{aligned}$$

2.1 Homogeneous-capacity case

From now on, we will consider a randomized on-line algorithm running against an oblivious adversary. Let's specialize, for now, to the case that all buckets have capacity $1 + \varepsilon$, hence the total load in each round is bounded above by m. We will analyze the following greedy algorithm.

ALGORITHM 2.2. *For each item i, choose a permutation π_i of the set of buckets, independently and uniformly at random from the set of all such permutations. In each round, items are assigned one at a time, in order from lightest to heaviest. Item i assigns all of its load to the first bucket in the permutation π_i, assuming this bucket has enough residual capacity to accept the load. Otherwise item i fills this bucket and moves to the next one in the permutation, continuing in this way until all of its load has been assigned.*

A subtle point in the design of this algorithm is that it is necessary, in each round, to order the items from lightest to heaviest and to assign them in that order. If the heavier items are assigned first, the number of full buckets will grow too rapidly, so that later items will not get their first choice. By ordering the items from lightest to heaviest, we ensure that only a small subset of the items fail to get their first-choice bucket; in fact, the analysis will show that this subset has expected size $O(m)$ in each round.

THEOREM 2.3. *Suppose each bucket has capacity* $1+\varepsilon$ *and the total load in each round is at most* m. *Given any sequence of load distributions*

$$\{\mu_{is} : 1 \le i \le n, 1 \le s \le t\},$$

the greedy algorithm achieves expected spread $O(\varepsilon^{-3}tm)$.

PROOF. Let "stage (i,s)" refer to the step in round s in which the greedy algorithm assigns item i, and let $F(i,s)$ denote the number of full buckets at the start of this step. We will reduce the problem of bounding expected spread to bounding each of the random variables $F(i,s)$ separately. The requisite bounds will then be stated and proved in a separate claim.

The link between estimating spread and counting full buckets can be understood as follows. Each time item i assigns positive load to bucket j, it may be classified into one of the following three cases:

1. $j = \pi_i(1)$. This does not contribute to the spread, since the definition of spread allows each item to use one bucket for free.

2. j's capacity is saturated by i in round s. Since each bucket is saturated at most once per round, these cases combine to contribute at most mt to the spread.

3. $j \neq \pi_i(1)$, and j is partially filled by i in round s.

For the purpose of establishing a bound of the form $Spread = O(mt)$, it suffices to consider only the contribution coming from case 3, since we have already argued that the contributions from cases 1 and 2 are $O(mt)$. Next, note that when item i assigns its load according to the greedy algorithm, it saturates all but at most one of the buckets it assigns its load to; thus it contributes at most one unit of spread to case 3. Moreover, if i saturates $\pi_i(1)$ and partly fills j in round s, then saturating $\pi_i(1)$ generates two "free credits" (since it satisfies cases 1 and 2 simultaneously) and one of these may be applied toward bucket j; thus we need only consider cases where an item *skips* the first bucket in its permutation and partly fills a later one.

The preceding paragraph establishes that an upper bound on spread can be obtained by counting the total number of times any item skips over the first bucket in its permutation. But the probability that this happens in stage (i,s) is exactly $F(i,s)/m$, so bounding the random variables $F(i,s)/m$ translates directly into bounding the spread.

To make the above argument precise, for each triple (i,j,s) with $1 \le i \le n, 1 \le j \le m, 1 \le s \le t$, define random variables Y_{ijs}, Z_{ijs} by

$$Y_{ijs} = \begin{cases} 1 & \text{if } i \text{ assigns positive load to } \pi_i(j) \\ & \text{in round } s, \\ 0 & \text{otherwise} \end{cases}$$

$$Z_{ijs} = \begin{cases} 1 & \text{if } i \text{ saturates } \pi_i(j) \text{ in round } s, \\ 0 & \text{otherwise} \end{cases}$$

There are some obvious inequalities which follow from the definitions of spread and the greedy algorithm. First,

$$Spread \le \sum_{s=1}^{t}\sum_{i=1}^{n}\sum_{j=2}^{m} Y_{ijs} \tag{1}$$

because we can obtain an upper bound for spread by charging item i one unit each time it assigns positive load to a bucket other than $\pi_i(1)$. Next,

$$\sum_{j=1}^{m} Y_{ijs} \le 1 + \sum_{j=1}^{m} Z_{ijs} \quad \forall i, s \tag{2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} Z_{ijs} \le m \quad \forall s. \tag{3}$$

The first inequality expresses the fact that the greedy algorithm fills all but at most one of the buckets it uses in stage (i,s), while the second expresses the fact that each bucket fills up at most once per assignment round.

Let $X_{is} = 1 - Y_{i1s}$ be the indicator random variable for the event that bucket $\pi_i(1)$ is full at the start of stage (i,s). Note that

$$\mathbf{E}(X_{is} \| F(i,s)) = F(i,s)/m$$

hence

$$\mathbf{E}(X_{is}) = E(F(i,s)/m)$$

We may rewrite inequality (2) as

$$\sum_{j=2}^{m} Y_{ijs} \le (1 - Y_{i1s}) + \sum_{j=1}^{m} Z_{ijs} = X_{is} + \sum_{j=1}^{m} Z_{ijs}$$

and combine it with (1) and (3) to obtain

$$\begin{aligned} Spread &\le \sum_{s=1}^{t}\sum_{i=1}^{n} X_{is} + \sum_{s=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m} Z_{ijs} \\ &\le \left(\sum_{s=1}^{t}\sum_{i=1}^{n} X_{is}\right) + mt \qquad (4) \\ \mathbf{E}(Spread) &\le \left(\sum_{s=1}^{t}\sum_{i=1}^{n} \mathbf{E}(F(i,s)/m)\right) + mt. \qquad (5) \end{aligned}$$

Now fix a round s and assume without loss of generality that the items are numbered in such a way that

$$\mu_{1s} \ge \mu_{2s} \ge \cdots \mu_{ns}, \tag{6}$$

i.e. that the greedy algorithm in round s considers the items in *reverse* order. We choose this numbering convention because of the following useful fact, which we will use repeatedly.

OBSERVATION 2.4. *If items $1, \ldots, n$ are numbered so that (6) holds, then*

$$\mu_{is} \leq m/i. \tag{7}$$

PROOF. Recall that $\sum_{k=1}^{n} \mu_{ks} \leq m$. By (6), the first i terms of this sum are all greater than or equal to μ_{is}, hence $\mu_{is} \leq m/i$. □

To bound $\mathbf{E}(F(i,s)/m)$, we will apply the following estimate, whose proof is deferred.

CLAIM 2.5. *If $r < \varepsilon/5$ and $i > 5\varepsilon^{-1}m$, then*

$$\Pr\left(\frac{F(i,s)}{m} > r\right) < e^{\left(1-\frac{\varepsilon^2 r}{12\mu_{is}}\right)m}. \tag{8}$$

Using (8), we may now compute the following bound on $\mathbf{E}(F(i,s)/m)$ for $i > 5\varepsilon^{-1}m$:

$$\begin{aligned}\mathbf{E}\left(\frac{F(i,s)}{m}\right) &= \int_0^1 \Pr\left(\frac{F(i,s)}{m} > r\right) dr \\ &< \int_0^1 \min\left(1, \exp\left(\left(1 - \frac{\varepsilon^2 \min(r, \varepsilon/5)}{12\mu_{is}}\right) m\right)\right) dr\end{aligned}$$

When $i > 60\varepsilon^{-3}m$, so that $12\varepsilon^{-2}\mu_{is} < \varepsilon/5$, we may divide the domain of integration into three subintervals $[0, 12\varepsilon^{-2}\mu_{is}]$, $[12\varepsilon^{-2}\mu_{is}, \varepsilon/5]$, $[\varepsilon/5, 1]$, and let I_1, I_2, I_3 denote the integrals over these three subintervals, respectively. Then

$$\begin{aligned} I_1 &= \int_0^{12\varepsilon^{-2}\mu_{is}} dr \\ &= 12\varepsilon^{-2}\mu_{is} \\ I_2 &< \int_{12\varepsilon^{-2}\mu_{is}}^{\infty} \exp\left(\left(1 - \frac{\varepsilon^2 r}{12\mu_{is}}\right) m\right) dr \\ &\leq 12\varepsilon^{-2}\mu_{is} \\ I_3 &< \int_0^1 \exp\left(\left(1 - \frac{\varepsilon^3}{60\mu_{is}}\right) m\right) dr \\ &\leq \exp\left(m - \frac{\varepsilon^3}{60} i\right)\end{aligned}$$

Summing over $i = \lceil 60\varepsilon^{-3}m \rceil, \ldots, n$, the contributions from I_1, I_2 are both bounded by $\sum_{i=1}^{n} 12\varepsilon^{-2}\mu_{is} = 12\varepsilon^{-2}m$, while the contribution from I_3 is bounded by

$$\sum_{j=0}^{\infty} e^{-\frac{\varepsilon^3}{60}j} < 1 + 60\varepsilon^{-3}.$$

We conclude that

$$\begin{aligned}\sum_{i=1}^{n} \mathbf{E}(F(i,s)/m) &= \sum_{i \leq 60\varepsilon^{-3}m} \mathbf{E}(F(i,s)/m) \\ &\quad + \sum_{i > 60\varepsilon^{-3}m} \mathbf{E}(F(i,s)/m) \\ &< 60\varepsilon^{-3}m + 24\varepsilon^{-2}m + 1 + 60\varepsilon^{-3} \\ &= O(\varepsilon^{-3}m)\end{aligned}$$

Plugging this into (5) yields

$$\begin{aligned}\mathbf{E}(\mathit{Spread}) &\leq \left(\sum_{s=1}^{t}\sum_{i=1}^{n} \mathbf{E}(F(i,s)/m)\right) + mt \\ &= O(\varepsilon^{-3}mt) + mt\end{aligned}$$

from which the theorem follows.

Applying Markov's Inequality to Theorem 2.3, we may convert our upper bound on expected spread into an upper bound on spread that holds with high probability.

COROLLARY 2.6. *With probability $1-\delta$, the greedy algorithm achieves spread $O(\varepsilon^{-3}\delta^{-1}tm)$.*

The same analysis can be adapted to give a spread bound which is logarithmic instead of linear in $1/\delta$, at the cost of a slightly worse dependence on t. Here and throughout the paper, we denote the natural logarithm function by "log".

THEOREM 2.7. *With probability $1-\delta$, the greedy algorithm achieves spread $O(\varepsilon^{-3}(mt\log(1/\varepsilon) + t\log(t/\delta))$.*

PROOF SKETCH. Based on Claim 2.5, we know that $\mathbf{E}(F(i,s)/m) = O(\mu_{is})$, and that $\Pr(F(i,s)/m > \alpha\mu_{is})$ is exponentially small in α. This exponential tail bound allows us to derive a sharper tail bound on spread than the one obtained in Corollary 2.6 using Markov's Inequality. We will specify a constant α, depending only logarithmically on t and $1/\delta$, such that with probability $1-\delta/2$ or better, *none* of the random variables $F(i,s)/m$ (for $i > 5\varepsilon^{-1}m$) exceeds $\alpha\mu_{is}$. Conditional on this high-probability event, we will prove that the spread is linear in αmt with high probability, thereby establishing the desired bound on spread.

Put

$$\alpha = 24\varepsilon^{-2}\left(3 + \frac{1}{m}\log(\frac{2t}{\delta}) + \log(12\varepsilon^{-2})\right).$$

A tedious calculation verifies that

$$\alpha > 12\varepsilon^{-2}\left(1 + \frac{1}{m}\log(2\alpha m^2 t/\delta)\right),$$

hence

$$\begin{aligned}\Pr\left(\frac{F(i,s)}{m} > \alpha\mu_{is}\right) &< e^{(1-\frac{\varepsilon^2}{12}\alpha)m} \\ &< e^{-\log(2\alpha m^2 t/\delta)} \\ &= \frac{\delta}{2\alpha m^2 t}\end{aligned}$$

when $\alpha\mu_{is} < \varepsilon/5$. By the union bound, with probability at least $1-\delta/2$ the inequality

$$F(i,s)/m \leq \alpha\mu_{is}$$

holds for all (i,s) with $1 \leq s \leq t$, $5\varepsilon^{-1}\alpha m \leq i \leq \alpha m^2$. Moreover, when $i = \lceil \alpha m^2 \rceil$, we have $\alpha\mu_{is} \leq 1/m$, so $F(i,s)/m < 1/m$ with high probability. By the integrality of $F(i,s)$, this actually implies $F(i,s) = 0$,

i.e. no buckets fill up before stage $(\alpha m^2, s)$ in round s. So with probability at least $1 - \delta/2$,

$$\sum_{i=1}^{n} F(i,s)/m < (5\varepsilon^{-1} + 1)\alpha m \quad \forall s.$$

Fixing s and conditioning on any particular set of values for the random variables $\{F(i,s)\}_{i=1}^{n}$, the Bernoulli random variables X_{is} are independent with expected sum $\sum_{i=1}^{n} F(i,s)/m$. By the Chernoff bound in Lemma 2.8,

$$\Pr\left(\sum_{i=1}^{n} X_{is} > 8\varepsilon^{-1}\alpha m \,\middle\|\, \sum_{i=1}^{n} F(i,s)/m < 4\varepsilon^{-1}\alpha m\right) < e^{-\frac{4}{3\varepsilon}\alpha m} < \delta/2t$$

so another application of the union bound over t assures that the probability that $\exists s \ \sum_{i=1}^{n} X_{is} > 8\varepsilon^{-1}\alpha m$ is less than $\delta/2$. But (4) now tells us that the spread is $O(\varepsilon^{-1}\alpha m t)$ with probability at least $1-\delta$. Substituting for α, we recover the bound claimed in the theorem. □

We believe that it should be possible to simultaneously strengthen Theorems 2.3 and 2.7 to obtain the bound $\mathit{Spread} = O(\varepsilon^{-3}\log(\delta^{-1})tm)$ with probability $1-\delta$. However, the proof may become much more elaborate, because it will be necessary to account for the possible dependence between the spread in different assignment rounds.

2.2 Proof of Claim 2.5

We now turn to proving Claim 2.5, which was deferred in the proof of Theorem 2.3. The intuition behind the proof is this: within a fixed round s, if the random variables measuring the amount of load assigned to a set S of buckets in different stages of the greedy algorithm were *independent*, then we could apply Chernoff bounds to derive an exponential tail bound resembling the one stated in Claim 2.5. However, these random variables are not independent because as buckets start to fill up, it influences the probability that load in future stages will be assigned to buckets in S. Thus, a subtler argument is required, capturing the intuition that if only a very small fraction of the buckets are full, the dependence between the random variables is very weak and should not upset the tail bound too much.

We will need the following lemma, which generalizes the classical Chernoff bound as found, for instance, in [8]. The lemma deals with cases where there is a limited amount of dependence between the random variables. The proof is standard, and is given here for the purpose of making the exposition self-contained.

LEMMA 2.8. *Let $X_1, \ldots, X_n$ be a sequence of random variables taking values in the interval $[0, c]$, and let $\{c_i\}_{i=1}^{n}$ be constants such that*

$$E(X_i \| X_1, X_2, \ldots, X_{i-1}) \le c_i$$

for all i and for all possible values of $(X_1, \ldots, X_{i-1})$. If $M = \sum_{i=1}^{n} c_i$, then for $0 < \varepsilon \le 1$,

$$\Pr\left(\sum_{i=1}^{n} X_i > (1+\varepsilon)M\right) < e^{-\frac{1}{3}\varepsilon^2 M/c}.$$

PROOF. The bounds $0 \le X_i \le c$ and

$$E(X_i \| X_1, X_2, \ldots, X_{i-1}) \le c_i,$$

together with Jensen's inequality applied to the convex funcion e^{tx}, imply the following bound on the conditional moment generating function of X_i:

$$\begin{aligned} E(e^{tX_i} \| X_1, X_2, \ldots, X_{i-1}) &\le 1 - \frac{c_i}{c} + \frac{c_i}{c}e^{ct} \\ &\le \exp\left(\frac{c_i}{c}(-1 + e^{ct})\right) \end{aligned}$$

Define partial-sum random variables $Y_k = \sum_{i=1}^{k} X_i$. We have

$$\begin{aligned} E(e^{tX}) &= E(e^{tY_n}) \\ &= E(e^{tX_n} e^{tY_{n-1}}) \\ &= E(E(e^{tX_n} \| X_1, \ldots, X_{n-1}) e^{tY_{n-1}}) \\ &\le \exp\left(\frac{c_n}{c}(-1 + e^{ct})\right) E(e^{tY_{n-1}}) \end{aligned}$$

By induction, we obtain

$$E(e^{tX}) < \exp\left(\frac{-1 + e^{ct}}{c} M\right),$$

and an application of Markov's inequality gives

$$\Pr(e^{tX} > e^{t(1+\varepsilon)M}) < \exp\left(\frac{-1 + e^{ct}}{c} M - t(1+\varepsilon)M\right)$$

Setting $t = \frac{1}{c}\log(1+\varepsilon)$ in the above inequality produces the bound

$$\Pr(X > (1+\varepsilon)M) < \exp((\varepsilon - (1+\varepsilon)\log(1+\varepsilon))M/c).$$

The lemma follows upon observing that

$$\varepsilon - (1+\varepsilon)\log(1+\varepsilon) < -\frac{1}{3}\varepsilon^2$$

for $0 < \varepsilon \le 1$. □

PROOF OF CLAIM 2.5. As stated above, the general idea is to bound the probability that too many buckets fill up, using the Chernoff bound (Lemma 2.8). It is tricky to deal with the greedy algorithm directly, since the random variables measuring the amount of load assigned to a set of buckets by different items are not independent. Instead we will compare the greedy algorithm to an alternative assignment process where the load variables are dominated by sums of nearly-independent random variables.

Consider the following three assignment processes, each of which defines load random variables $L(i,j,s)$ and "full-bucket-counting" random variables $F(i,s)$.

Greedy algorithm The algorithm is defined in section 2.1. At stage (i,s), $L(i,j,s)$ is the load on bucket j and $F(i,s)$ is the number of full buckets.

Greedy unsplittable assignment At stage (i,s), define $A(i,s)$ to be the set of buckets whose residual capacity is at least μ_{is}. If $A(i,s)$ is non-empty, assign item i to the element of $A(i,s)$ which appears earliest in π_i. Else assign item i as in the greedy algorithm. $L'(i,j,s)$ is the load on bucket j at stage (i,s), and $F'(i,s) = m - |A(i,s)|$.

Reckless unsplittable assignment At stage (i,s), choose an arbitrary set $B(i,s)$ of buckets subject to the following constraints:

- $|B(i,s)| \geq (1-r)m$.
- $B(i,s)$ contains the set of all buckets whose residual capacity is at least μ_{is}, and is equal to that set if its cardinality is greater than or equal to $(1-r)m$.

Assign item i to the element of $B(i,s)$ which appears earliest in π_i, without regard for whether this assignment will overload the selected bucket or not. $L''(i,j,s)$ is the load on bucket j at stage (i,s), and $F''(i,s)$ is the number of buckets with residual capacity less than μ_{is} at stage (i,s).

The following lemma presents some simple facts about the relations between these assignment processes.

LEMMA 2.9. *The assignment processes defined above satisfy*

1. *For all $j \in A(i,s)$, $L(i,j,s) \leq L'(i,j,s)$.*
2. *$F(i,s) \leq F'(i,s)$ for all (i,s).*
3. *If $F''(i,s) \leq rm$, then $L''(i,j,s) = L'(i,j,s)$ for all j, and $F''(i,s) = F'(i,s)$.*

PROOF. Property 1 is proved by induction on items i, ordered from lightest to heaviest. (This is the order in which the items are assigned by all three assignment processes; however, according to our numbering convention, it is actually a downward induction starting from $i = n$.) In the base case, $L(i,j,s) = L'(i,j,s) = 0$ and there is nothing to prove. For the induction step assume $L(i+1,j,s) \leq L'(i+1,j,s)\ \forall j \in A(i+1,s)$. This inequality holds *a fortiori* for all $j \in A(i,s)$ since $A(i,s) \subseteq A(i+1,s)$. (To be included in $A(i,s)$ as opposed to $A(i+1,s)$, a bucket must have a *greater* amount of residual capacity at a *later* stage of the assignment process. This is a stricter requirement, so $A(i,s) \subseteq A(i+1,s)$.) Now in stage (i,s), greedy unsplittable heaps all the load of item i on the element j of $A(i,s)$ which appears earliest in π_i. For this bucket, L' grows by μ_{is} while L grows by at most μ_{is}, so the induction hypothesis is preserved. For all other buckets in $A(i,s)$, neither greedy nor greedy-unsplittable assigns *any* load to them in stage (i,s), the reason being that bucket j has enough residual capacity to absorb all the load of item i. (This follows by the definition of j in the greedy-unsplittable case; therefore by the induction hypothesis it holds in the greedy case as well.)

Property 2 is a trivial consequence of property 1, and of the definition of $F(i,s)$ and $F'(i,s)$.

Property 3 is again proved by downward induction on i, i.e. considering items in the order they are treated by these assignment processes. The inductive hypothesis is that greedy-unsplittable and reckless-unsplittable make all the same decisions from the start (stage n) up to stage i. In the base case this is trivial. In the induction step, one observes that $F''(i,s)$ grows as the assignment process proceeds. (The residual capacity of buckets is shrinking, and μ_{is} is growing, so the number of buckets with residual capacity below μ_{is} grows.) Thus the hypothesis $F''(i,s) \leq rm$ implies that the same inequality held in all prior rounds as well. We now apply the induction hypothesis to conclude that the greedy-unsplittable and reckless-unsplittable processes made all the same decisions up to stage i. If so, then $B(i,s) = A(i,s)$ since $B(i,s)$ is defined by the same criterion as $A(i,s)$ as long as $F''(i,s) \leq rm$. But if $B(i,s) = A(i,s)$, then greedy-unsplittable and reckless-unsplittable will make the same decision in stage (i,s), which verifies the induction step. □

Fix a set S of rm buckets, and let X_i denote the random variable measuring the amount of load assigned to buckets in S during stage (i,s) of the reckless unsplittable assignment. Note that $0 \leq X_i \leq \mu_{is}$ and

$$\begin{aligned} E(X_i \| X_n, X_{n-1}, \ldots, X_{i+1}) &= E(X_i \| B(i,s)) \\ &= \frac{rm}{|B(i,s)|}\mu_{is} \\ &< \frac{r}{1-r}\mu_{is}. \end{aligned}$$

In order for S to be contained in $F''(i,s)$, it must be the case that each of the rm buckets in S has load at least $1+\varepsilon-\mu_{is}$ at stage (i,s), which would imply that

$$\sum_{k=i}^{n} X_k > (1+\varepsilon-\mu_{is})rm.$$

We may bound the probability of this event by applying Lemma 2.8 to the sequence of random variables $X_n, X_{n-1}, \ldots, X_i$. The lemma refers to a parameter $M = \sum_{i=1}^{n} c_i$ which in our case is

$$M = r/(1-r)\sum_{k=i}^{n} \mu_{ks} < rm/(1-r).$$

Note that

$$\begin{aligned} (1+\varepsilon-\mu_{is})rm &> (1+\varepsilon-\mu_{is})(1-r)M \\ &> (1+\frac{4}{5}\varepsilon)(1-\frac{1}{5}\varepsilon)M \\ &= (1+\frac{3}{5}\varepsilon-\frac{4}{25}\varepsilon^2)M \\ &> (1+\frac{3}{5}\varepsilon-\frac{2}{25}\varepsilon)M \\ &> (1+\frac{1}{2}\varepsilon)M \end{aligned}$$

where we have used the facts that $\varepsilon < 1/2$ (by assumption), $r < \varepsilon/5$ (by assumption), and $\mu_{is} < \varepsilon/5$ (by the assumption $i > 5\varepsilon^{-1}m$, combined with observation 2.4). Putting all this together, we apply Lemma 2.8 to conclude that

$$\Pr\left(\sum_{k=i}^{n} X_k > (1+\varepsilon-\mu_{is})rm\right) < e^{-\frac{1}{12}\varepsilon^2 rm/\mu_{is}}. \quad (9)$$

The left side of (9) is an upper bound on the probability that S is contained in the set $F''(i,s)$. The union bound

now gives

$$\begin{aligned}\Pr(F''(i,s) > rm) &< \binom{m}{rm} e^{-\frac{1}{12}\varepsilon^2 rm/\mu_{is}} \\ &< 2^m e^{-\frac{1}{12}\varepsilon^2 rm/\mu_{is}} \\ &< e^{\left(1-\frac{\varepsilon^2 r}{12\mu_{is}}\right)m}.\end{aligned}$$

But if $F''(i,s) \le rm$ then $F(i,s) \le rm$ as well (by parts (2) and (3) of Lemma 2.9), so the claim is proved.

2.3 Heterogeneous-capacity case

We deal now with the case where the buckets have differing capacities.

THEOREM 2.10. *Suppose the total capacity of the buckets exceeds the total load in each round by a factor of at least $1+\varepsilon$. There is a randomized on-line algorithm which achieves expected spread $O(\varepsilon^{-4}tm)$ on any sequence of load distributions $\{\mu_{is}\}$*

PROOF. If the capacities $\rho_1, \ldots, \rho_m$ are all common multiples of some number z, then we may split bucket j into j/z "virtual buckets" and run the greedy algorithm on these virtual buckets. The total number of such buckets is ρ/z, where $\rho = \sum_{j=1}^m \rho_j$ is the total capacity. By Theorem 2.3 the expected spread is $O(\varepsilon^{-3}t\rho/z)$.

In the general case, taking $z = \rho\varepsilon/2m$, we may round down each ρ_j to the nearest multiple of z, i.e. replace ρ_j with $z\lfloor \rho_j/z \rfloor$ and modify the algorithm to ignore the excess capacity. In doing so, we lose at most z units of capacity per server, for a total of $zm = \rho\varepsilon/2$ lost units of capacity. But the capacity in each round exceeds the load by $\rho\varepsilon$, so the total of the revised capacities still exceeds the load in each round by a factor of $1 + \varepsilon/2$. Thus the expected spread will be $O(\varepsilon^{-3}t(2m/\varepsilon)) = O(\varepsilon^{-4}tm)$. □

2.4 Extension to multiple variables

The spread-minimization problem admits a natural generalization to the multi-variable setting, in which loads and capacities take values in $\mathbb{R}^d$. (In the applications, the variables typically represent different server resources, e.g. bandwidth and CPU.) An instance of the problem now consists of nt load vectors $\{\vec{\mu}_{is}\}$, whose k-th component will be denoted μ_{isk}, and m capacity vectors $\{\vec{\rho_j}\}$. As before, the task is to fractionally assign items to buckets in each of t rounds, in such a way that the capacity constraints in each load variable are satisfied. Spread is defined exactly as before.

It is trivial to prove a lower-bound of $dt(m-1)$ on the worst-case spread povided $n \ge dt$. Simply pick disjoint sets of d items, one for each of the t rounds; let $\{i(1,s), i(2,s), \ldots, i(d,s)\}$ denote the set of items chosen for round s. Put load $m - 1 + 1/n$ in the k-th component of $\vec{\mu}_{i(k,s),s}$ for $1 \le k \le d$, $1 \le s \le t$, setting the load to $1/n$ in all other components of all load vectors. Then, assuming each server has capacity $(1, 1, \ldots, 1)$, in each round each of the d chosen items must spread to all servers, resulting in an unnormalized spread of $n + dt(m-1)$. If instead each server has capacity $(1+\varepsilon, \ldots, 1+\varepsilon)$, then in each round each of the d chosen items must spread to $m/(1+\varepsilon)$ servers, resulting in an unnormalized spread of $n + dt[m/(1+\varepsilon) - 1]$.

In considering on-line algorithms, we will assume as before that capacity exceeds load by a factor of at least $1+\varepsilon$ in each round (and in each load variable). The greedy algorithm studied above is not suitable in the multi-variable case, even when the buckets are equicapacitated. The reason is that when buckets reach saturation in one or more of their load coordinates, they are left with a nonzero residual capacity vector which is now off-limits to the algorithm. In this way, it is possible for the greedy algorithm to become wedged in a state where it is impossible to find a feasible solution without revoking earlier decisions. Moreover, such scenarios cannot be relegated to the realm of "small probability"; it is easy to come up with examples of problem instances where the greedy algorithm's probability of getting wedged is arbitrarily close to 1.

In light of this complication, we are forced to look at non-greedy algorithms. The greedy algorithm gets into trouble when large items saturate a bucket in one or more load coordinates. Consequently, it makes sense to consider an algorithm which splits large items into smaller pieces so as to ensure that, with high probability, no bucket ever gets saturated in *any* load coordinate. The following algorithm divides large items up into equal-sized pieces; the number of such pieces is determined by the largest component of the item's load vector. The analysis will show that the spread is within a log-factor of the optimal worst-case spread, with high probability, in the equicapacitated case. Assume that the capacity of each bucket is $(1+\varepsilon, 1+\varepsilon, \ldots, 1+\varepsilon)$ and that the load in each round is bounded above by $(m, m, \ldots, m)$.

ALGORITHM 2.11. *Let*

$$J(i,s) = \max_{1 \le k \le d} \left\lceil 3\varepsilon^{-2} \log\left(\frac{dtm}{\delta}\right) \mu_{isk} \right\rceil.$$

In round s, let $\mathcal{A}_s$ denote the fractional assignment which distributes the load on each item i evenly among the buckets $\{\pi_i(1), \pi_i(2), \ldots, \pi_i(J(i,s))\}$, where this set is interpreted as the set of all m buckets if $J(i,s) \ge m$. If $\mathcal{A}_s$ satisfies all the capacity constraints, use it. Otherwise pick an arbitrary assignment which satisfies the capacity constraints.

THEOREM 2.12. *With probability $1-\delta$, the above algorithm uses the assignment $\mathcal{A}_s$ in each round s. If so, the spread is $O(\varepsilon^{-2}dtm\log(dtm/\delta))$.*

PROOF. To bound the probability that $\mathcal{A}_s$ overloads the k-th load variable in bucket j, we use the Chernoff bound (Lemma 2.8). Fix j, k, s, and let X_i denote the k-th component of the load vector received by bucket j in stage (i,s). The random variables $\{X_i\}_{i=1}^n$ are mutually independent and have distributions given by

$$X_i = \begin{cases} \mu_{isk}/J(i,s) & \text{with probability } J(i,s)/m \\ 0 & \text{with probability } 1 - J(i,s)/m \end{cases}$$

It follows that

$$EX_i = \mu_{isk}/m$$

and X_i is either deterministically equal to μ_{isk}/m (in the case $J(i,s) = m$) or else satisfies the bound

$$0 \leq X_i \leq \frac{\mu_{isk}}{J(i,s)} \leq \frac{1}{3}\varepsilon^2 \left(\log\left(\frac{dtm}{\delta}\right)\right)^{-1}$$

The sum $X = \sum_{i=1}^{n} X_i$ satisfies $EX = \sum_{i=1}^{n} \mu_{isk}/m \leq 1$, and the Chernoff bound yields

$$\begin{aligned} \Pr(X > 1+\varepsilon) &< \exp\left(\frac{-\varepsilon^2/3}{\frac{1}{3}\varepsilon^2\left(\log\left(\frac{dtm}{\delta}\right)\right)^{-1}}\right) \\ &= \exp\left(-\log\left(\frac{dtm}{\delta}\right)\right) \\ &= \frac{\delta}{dtm} \end{aligned}$$

Taking the union bound over all choices of k, s, and j, we see that the probability of any bucket getting overloaded in any round is at most δ, as claimed.

Finally, we must bound the spread, assuming that the algorithm chooses assignment A_s in each round s.

$$\begin{aligned} \mathit{Spread} &\leq \sum_{s=1}^{t}\sum_{i=1}^{n}(J(i,s)-1) \\ &\leq \sum_{s=1}^{t}\sum_{i=1}^{n}\max_{1\leq k\leq d}\left\lfloor 3\varepsilon^{-2}\log\left(\frac{dtm}{\delta}\right)\mu_{isk}\right\rfloor \\ &\leq \sum_{s=1}^{t}\sum_{i=1}^{n}\sum_{k=1}^{d}\left\lfloor 3\varepsilon^{-2}\log\left(\frac{dtm}{\delta}\right)\mu_{isk}\right\rfloor \\ &\leq 3\varepsilon^{-2}\log\left(\frac{dtm}{\delta}\right)\sum_{s=1}^{t}\sum_{k=1}^{d}\sum_{i=1}^{n}\mu_{isk} \\ &\leq 3\varepsilon^{-2}\log\left(\frac{dtm}{\delta}\right)dtm \end{aligned}$$

as claimed. □

3. OFF-LINE ALGORITHMS

As explained in Section 2.4, there is a lower bound of $dt(m-1)$ on the worst-case spread of any algorithm. We will present a simple algorithm which exactly achieves worst-case spread $dt(m-1)$. The algorithm is based purely on linear algebra, and is inspired by the set-balancing algorithm of Beck-Fiala [1]. As mentioned in the introduction, the same ideas underlie the rounding techniques of Lenstra, Shmoys and Tardos [7], [10].

PROPOSITION 3.1. *Consider a linear system $\mathbf{Ax} = \mathbf{b}$ of p equations in q unknowns. Let $\mathbb{R}^q_+$ denote the subset of $\mathbb{R}^q$ consisting of points whose coordinates are non-negative. If the linear system has a solution $\mathbf{x} \in \mathbb{R}^q_+$, then there is a solution $\mathbf{x}' \in \mathbb{R}^q_+$ such that at most p coordinates of $\mathbf{x}'$ are non-zero.*

PROOF. Intuitively, the solution set of the linear system has at least $q-p$ degrees of freedom, which should allow us to set at least $q-p$ of the coordinates to zero, leaving at most p non-zero coordinates. (In the language of linear programming, this corresponds to finding a basic feasible solution of the linear program $\mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq 0$.) The following paragraphs make this argument precise.

By assumption, the solution set of $\mathbf{Ax} = \mathbf{b}$ has non-empty intersection with $\mathbb{R}^q_+$. Among the intersection points, let $\mathbf{x}'$ be one with as few non-zero coordinates as possible. Assume without loss of generality that the first r coordinates of $\mathbf{x}'$ are non-zero; we must show $r \leq p$. Let $\mathbb{R}^r \subseteq \mathbb{R}^q$ denote the subspace consisting of points which are zero in all but the first r coordinates. Restricting our linear system to $\mathbb{R}^r$, we obtain a new linear system $\mathbf{A'y} = \mathbf{b'}$ with a solution in the interior of $\mathbb{R}^r_+$. If $r > p$, then the solution set of $\mathbf{A'y} = \mathbf{b'}$ is a positive-dimensional affine subspace of $\mathbb{R}^r$ intersecting $\mathbb{R}^r_+$. But then it must intersect the boundary of $\mathbb{R}^r_+$, yielding a solution with fewer non-zero coordinates. This contradicts the minimality of r. □

THEOREM 3.2. *For the off-line spread-minimization problem, there is a polynomial-time algorithm achieving worst-case spread $t(m-1)$.*

PROOF. For simplicity, assume all buckets have equal capacity. (We will indicate later how to eliminate this assumption.) As in the introduction, let x_{ijs} denote the fraction of load on item i assigned to bucket j in round s. We will find an assignment where x_{ijs} is independent of s, and where each bucket receives exactly $1/m$ of the load in each round. This means that we seek real numbers x_{ij} satisfying

$$x_{ij} \geq 0 \qquad \forall i,j \tag{10}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \tag{11}$$

$$\sum_{i=1}^{n} x_{ij}\mu_{is} = \frac{1}{m}\sum_{i=1}^{n}\mu_{is} \qquad \forall j,s \tag{12}$$

Line 11 accounts for n equations, and line 12 accounts for mt equations. However, the equations (12) for $j = m$ are redundant, as they follow from (11) together with (12) for $j < m$. Omitting these redundant linear equations, we get $n + t(m-1)$ equations in mn unknowns. The solution $x_{ij} = 1/m\ \forall i,j$ illustrates that there exists a solution in $\mathbb{R}^{mn}_+$. Applying the proposition above, we obtain a solution in which at most $n + t(m-1)$ of the variables x_{ij} are positive. This solution achieves spread at most $t(m-1)$.

A polynomial-time algorithm to compute this solution $x_{ij} \in \mathbb{R}^{mn}$ starts at the point $(1/m, 1/m, \ldots, 1/m)$ and sets coordinates to zero one-by-one, while continuing to satisfy equations (11) and (12) at each step. This requires solving $O(mn)$ linear systems, each of size $O(m(n+t))$, so it runs in polynomial time as claimed.

If the buckets do not have equal capacities, the proof requires only minor modifications. Let $\rho = \sum_{j=1}^{m}\rho_j$ denote the total capacity of the buckets. Then equation 3 above should be replaced by

$$\sum_{i=1}^{n} x_{ij}\mu_{is} = \frac{\rho_j}{\rho}\sum_{i=1}^{n}\mu_{is} \qquad \forall j,s$$

and the initial interior-point solution is given by $x_{ij} =$

$\frac{\rho_j}{\rho}$ $\forall i, j$. The rest of the proof proceeds exactly as above. □

REMARK 3.3. *In the multivariate case, if the buckets have equal capacities, the same algorithm as above achieves spread $dt(m-1)$ The proof is exactly the same, except that the mt equations (12) are replaced by the dmt equations*

$$\sum_{i=1}^{n} x_{ij}\mu_{isk} = \frac{1}{m}\sum_{i=1}^{n}\mu_{isk} \qquad \forall j, s, k$$

However, the extension to the heterogeneous-capacity case fails because the existence of an interior solution of the linear program — or indeed, of any solution at all — is no longer guaranteed. For instance, it is infeasible to assign two items, each of size $(1/2, 1/2)$, to a pair of buckets whose capacities are $(1/3, 2/3)$ and $(2/3, 1/3)$.

4. FURTHER DIRECTIONS AND OPEN QUESTIONS

As we have indicated, some of the bounds proved above for on-line spread-minimizing algorithms are probably not optimal. In particular, the multi-varable case still has a gap between the lower bound of $\Omega(dtm)$ and the upper bound of $O(dtm \log dtm)$. We conjecture that the lower bound is tight. We also suspect that, in the single-variable case, the greedy algorithm achieves a spread depending logarithmically on $1/\delta$, not linearly.

Also interesting is the fact that we don't know how to handle heterogeneous capacities in the multi-variable case, except when all of the capacities are scalar multiples of each other. In fact, we have seen that with non-parallel capacity vectors, the existence of a feasible solution imposes non-trivial conditions on the problem instance. This makes it more difficult to guess what the proper conjecture should be in this setting.

We have not analyzed on-line spread-minimization algorithms from the standpoint of competitive ratio. Rather, we have compared the on-line algorithm's *worst-case* spread with that of the optimal off-line solution. In fact, it is not clear how one ought to define the competitive ratio of such algorithms. (If one compares the unnormalized spread with the optimal unnormalized spread, then all the algorithms presented above achieve competitive ratios arbitrarily close to 1 in the large-n limit. If one compares normalized spread with optimal normalized spread, then the possibility arises that the optimal normalized spread is zero; in such cases, is the on-line algorithm required to have zero spread?) However, there are clearly interesting questions about competitive ratio (e.g. in the case where n are tm are comparable in magnitude) and the answers may be non-trivial.

Throughout the paper, we have adopted a definition of spread based on counting measure. This definition is the correct one for the applications if all of the cached objects are of roughly similar size. However, to model cases where the sizes of the cached objects can vary widely, it is desirable to define the obvious weighted version of spread, and to seek algorithms which minimize this quantity. (Presumably the algorithms would be evaluated according to a suitably-defined notion of competitive ratio.) It is easy to see that the algorithms analyzed in this paper are not satisfactory in the weighted-spread context.

Finally, we anticipate further interesting work on the theme of consistent algorithms, i.e. those whose outputs don't change too much when presented with a sequence of varying inputs. So far, hashing ([5], [6]) and load-balancing have been studied from this perspective, but it is possible that analyzing other optimization problems (e.g. scheduling, network flow) from this standpoint would yield interesting and applicable results.

5. REFERENCES

[1] BECK, J. AND FIALA, T. "Integer-making" theorems, *Discrete Appl. Math.***3**, 1-8.

[2] BECK, J. AND SÓS, V. Discrepancy Theory. In *Handbook of Combinatorics, vol. II* (Ed. R. Graham, M. Grötschel, and L. Lovász), Elsevier Science, Amsterdam, 1995.

[3] COFFMAN, E., GAREY, M., AND JOHNSON, D. Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems*, D. Hochbaum, Ed. PWS Publishing, Boston 1997, 46-93.

[4] HALL, L. Approximation algorithms for scheduling. In *Approximation algorithms for NP-hard problems*, D. Hochbaum, Ed. PWS Publishing, Boston 1997, 1–46.

[5] KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN, D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (El Paso, TX, May 1997), pp. 654-663.

[6] LEWIN, D. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, Department of EECS, MIT, 1998. Available at the MIT Library, `http://thesis.mit.edu/`.

[7] LENSTRA, J., SHMOYS, D., AND TARDOS, E. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming* **46**, 259-271.

[8] MOTWANI, R. AND RAGHAVAN, P. *Randomized Algorithms* Cambridge University Press, New York, NY, 1995.

[9] SPENCER, J. Six standard deviations suffice, *Trans. Amer. Math. Soc.* **289**, 679-706.

[10] SHMOYS, D., AND TARDOS, E. An approximation algorithm for the generalized assignment problem, *Math. Programming* **62** 461-474