Florida International University Technical Report **TR-2012-10-01**

Vector Repacking Algorithms for Power-Aware Computing

Mario E. Consuegra, Giri Narasimhan, Raju Rangaswami
Florida International University

mcons004@fiu.edu {giri, raju}@cs.fiu.edu

Abstract

Many resource allocation problems can be cast as versions of the multi-dimensional vector packing problem. However, in most applications the inputs are inherently dynamic. In this paper we experiment with practical algorithms for the *vector repacking* problem and its variants. Vector repacking, like vector packing, aims to pack a set of input vectors such that the number of bins used is minimized, but has the additional goal of minimizing the changes from the previous packing. We also consider a variant of vector repacking that stores additional copies of items with the goal of improving the performance of vector repacking algorithms. We discuss the best versions of the *Repack* and *Replicas* algorithms to address the vector repacking problem and its variant. Experiments show that our algorithms result in marked improvements over existing heuristics (SRCMap) for energy-proportional storage management. In addition, our algorithms are paramterized so that they can be effectively optimized for a variety of resource allocation applications with different input characteristics and different cost functions.

1 Introduction

Power consumption by data centers has displayed phenomenal growth in recent years and is expected to grow at even faster rates [9, 8]. It is therefore imperative to find ways to make data center operations more energy efficient. Toward this goal, Barroso and Hozle [11] have proposed the approach of "energy proportionality". Most data centers are provisioned for peak performance, while average loads can be much more modest. An energy-proportional approach allows for power consumption to vary with the usage; they attempt to put as many servers as possible into low-energy consumption states and pack the maximum sustainable amount of work into each server kept active and in a high energy consumption state.

Resource allocation problems can be cast as vector packing problems by representing each item (task) and each bin (server) as a multi-dimensional vector with dimensions for relevant parameters. (We refer to the vector representing the item or bin as its *profile*.) In the storage systems application, relevant parameters of the profile include working set size, workload intensity (measured in IOPS), miss rate, etc. In the VM management application, the dimensions could represent the requirements of the VM for resources such as CPU, memory, bandwidth, etc. Energy-efficient computing can be achieved by considering the problem of (vector) packing these items into bins (servers) with the goal of optimizing (minimizing) the total power consumed by the servers. Server capacities in each of the cases directly translates to bin capacities for the abstracted multi-dimensional vector packing problem.

With the overall goal of optimizing the power consumption of data centers, Amur et al. [1], Verma et al. [17] and Thereska et al. [15] considered the problem of storing multiple replicas of data sets and working sets, while Panigrahy et al. [13], looked at the problem of efficiently packing virtual machines (VMs) with known static demands into servers with fixed capacitites. Using fewer bins (i.e., servers) directly translates to lower power consumption. Energy efficiency is thus achieved by keeping an optimal subset of servers in the system active while other servers are spun down or brought to a lower energy consumption state.

Vector Repacking: Given that workloads are inherently dynamic [11], we turn our attention to the challenging vector *repacking* problem. While good packings are useful, they may become sub-optimal as conditions change. Computing good packings from scratch may be prohibitively expensive because it may require all items to be moved. Here we cast the problems as a variant of the multi-dimensional vector packing where we are required to "repack" efficiently and where the cost of the resulting packing is also dependent on how it differs from the previous packing, the assumption being that repacking requires a costly "migration" of tasks, processes, or data. Our approaches are meaningful only when the vectors to be packed change their profiles relatively infrequently, thus making it worthwhile to reconsider the repacking of the entire set of tasks. The purely *dyanmic* bin packing problem (where profiles change incessantly) is also meaningful, but outside the scope of the present work.

Vector Repacking with Replicas: Next, we consider a practical variant of the vector repacking problem. In this variant, we assume that the system can store a limited number of extra copies (replicas) of select (or all) tasks, with the goal of reducing the cost of "migration". Here our experiments aim to *study the tradeoff between replication cost and the cost of task/data migration.*

Note that the problems considered here can also be cast in more general terms as combinatorial optimization problems, allowing for a host of other optimization approaches (e.g., ILP, semi-definite programming, rounding methods, etc.) to be applied. We do not consider these general approaches here, since they are unlikely to be as time-efficient as the algorithms evaluated here. Experiments and results for the two problems are described in Sections 3 and 4. The results show that using vector repacking is an effective and practical approach to deal with problems from the areas of dynamic resource allocation and power-aware computing. We show that allowing for extra copies (replicas) of the entities can be used with vector repacking approaches to find efficient solutions that attempt to minimize migration costs. We show results of our experiments with some real data sets (Section 5). Our conclusions are summarized in Section 6.

2 Related Work

The purpose of this section is to evaluate and select competitive (static) vector packing algorithms that could be useful to solve the dynamic repacking variants introduced above. As mentioned earlier, we model the problem of optimal placing of items such as working sets or VMs ("items") on disk servers or VM hosts ("bins") as a multi-dimensional vector packing problem. Efficient packings into the smallest number of bins translates to important energy savings. We assume that all bins have homogenous capacities in all dimensions and that the input is normalized such that the bins have a capacity of 1 in each dimension.

The multidimensional vector packing problem has been of interest for over three decades [12, 10] and many sophisticated approximation algorithms have been proposed for it. Even for $d = 1$, the vector packing problem (bin-packing) is NP-hard, and there are no approximation algorithms having an approximation ratio of $(\frac{3}{2} - \epsilon)$ for $\epsilon > 0$ unless $P = NP$ [3]. Hence finding efficient and practical algorithms to solve this problem is still a challenging task. For an excellent compilation of the relevant work on 1-dimensional bin packing the reader is referred to a survey by Coffman et al. [7].

For multi-dimensional vector packing Woeginger [18] ruled out an approximation scheme even for the case $d = 2$. For approximation algorithms, de la Vega and Lueker [6] obtained a $(d + \epsilon)$-approximation algorithm; and an improved algorithm by Bansal et al. [2] achieved an approximation ratio of $(\ln d + 1)$. Both algorithms are deemed not practical. For the 2-dimensional case, there is a $(\frac{1}{1-\rho})$-approximation algorithm (for any $\rho < 1$) called Hedging [5] (where ρ is the maximum length in any component for each item). Note that Hedging can be competitive and useful only for small ρ (e.g., $\rho < \frac{1}{2}$). Other relevant algorithms include generalizations of the most effective algorithms for the 1-dimensional case (e.g. GFFD), and were considered as candidates for the multi-dimensional vector problem considered here. Applications of vector packing to resource allocation problems have also been explored [13].

Note that the vector repacking problem has received less attention. In fact, we are not aware of any relevant work on the two variants considered in this paper.

We assume that the input to all variants of the vector packing problem includes a set of d-dimensional vectors within the unit d-dimensional cube, representing the d-dimensional profiles of n items that need to be packed. In other words, $\bar{v}_i = (v_{i1}, v_{i2}, \ldots, v_{id})$, where $v_{ij} \leq 1$ for $j = 1, \ldots, d$. The (static) vector packing problem is to partition S into a minimum set of subsets of S (bins), $\{S_1, S_2, \ldots, S_m\}$, such that $\sum_{\bar{v}_i \in S_k} \bar{v}_i \leq \bar{1}$, for $k = 1, \ldots, m$, where $\bar{1}$ is the d-dimensional vector of all 1's.

Based on prior work, there is strong evidence that *off-line* algorithms for (static) vector packing perform better (both theoretically and in practice) than their *on-line* counterparts. Thus for the 1-dimensional case, First-fit decreasing (FFD) performs better than First-fit (FF). Similar behavior has been observed for vector packing in higher dimensions. For practical vector packing in 1-dimensions, it is well known that FF and Best-fit (BF) [7] and their on-line counterparts, FFD and BFD, strike the best balance between their time complexity and performance in terms of number of bins. Stillwell et al. [14] showed FFDSum to be a good choice for resource allocation algorithms for virtualized service platforms. Generalziing BF and BFD for vector packing can be done in many ways. For $d \geq 2$, Panigrahy et al. [13] proposed an algorithm called FFD-EL2, which finds the bin with the closest L_2-distance between the vector profile of the item and the remaining space in the bin (represented as d-dimensional vector). Their experiments showed FFD-EL2 to be the most competitive among the vector packing algorithms. In summary, generalizations of FF and BF for higher dimensions seemed to be the best candidates for applying to the vector repacking problems. However, our experiments led to some

surprising results as shown below.

3 Vector Repacking

In practice, finding optimal or near-optimal placements of entities on servers is not the end of the story. When profiles of entities change, placements have to be modified, resulting in costly data migration between servers. Vector repacking is the problem that requires the simultaneous optimization of the number of bins as well as the amount of changes from a previous packing (i.e., migrations). The migration cost is modeled as a function of the difference between the previous packing and a new packing. More formally, we have:

Instance: A set of vectors $S = \{\bar{v}_1, \ldots, \bar{v}_n\}$, representing the d-dimensional profiles of the n items to be repacked, a partition of S, $B = \{S_1, \ldots, S_m\}$, representing the previous packing, and a cost function $f : (B, B') \rightarrow \mathbb{R}_{(0,1]}$ that assigns a cost to the change of packing from B to B'.

Problem: Find a packing $B' = \{S'_1, \ldots, S'_m\}$ such that $f(B, B')$ is minimized.

Note that we make the following assumptions. We assume that the cost function $f(B.B')$ is a combination of two costs – the cost of the packing B' (i.e., it depends on the number of bins in B') and the migration cost. We assume that the cost of migrating a specific item depends on its *size*, which is assumed to be one of the dimensions of vector profile representing the item (say, dimension 1). For example, the cost of migrating VMs is proportional to its memory footprint, while in storage systems, migration costs are proportional to the size of the working-set. We assume that the total migration cost for the whole repacking is simply the sum of the migration costs of individual items. In other words, other consequences of the migration (e.g., loss in computational time of a VM during the migration) are assumed to have minimal impact and negligble cost. Finally, we assume that the migration cost for a specific item depends only on the vector describing the item and not on the source or the destination of the move. For the applications in question, these assumptions are quite reasonable.

Applications: For resource allocation applications where one would like to assign tasks to servers, it is possible that a task may need to be migrated to a different server because its profile may change over a period of time. For example, a VM may become more or less compute intensive or memory intensive; in storage systems, a workload may have the miss rate characteristics change [17]. In the analogy of vector packing, it is possible that the bin may not be able to pack the same set of vectors as the parameters of the entities change, requiring migration of the entities from one bin (server) to another [16]. Here we run experiments simulating the scenario where a given set of data items is represented by a set of d-dimensional vectors. Our algorithms assume an initial packing for the items (by applying one of the static vector packing algorithms). Then as their profiles change the given placements may become untenable and new placements may be required, which involves data movement, whose cost is assumed to be proportional to the size of the migrated item. In the following experiments we study algorithms that aim to find a packing of data sets into a (approximately) minimum number of servers as the load varies over time while incurring a (approximately) minimum migration cost.

3.1 In Search of the Best Vector Repacking algorithms

The *Repack* Algorithm: For the vector repacking problem, the *naive* algorithm is to simply repack from scratch without using any information from the prior packing. Instead we propose a generalized heuristic, which we call *Repack* that uses the standard (static) vector packing algorithms as subroutines. This generalized heuristic for vector repacking has three stages and works as follows.

- The **first stage** involves vector eviction, where overflowed bins are identified and selected vectors are evicted.
- The **second stage** involves placement of the evicted vectors, which are packed either into one of the existing bins with adequate resources or into new bins.
- The **third and final stage** involves a packing reduction step, where we consider one bin at a time and check whether all its vectors can be placed in other open bins.

The last stage is to take care of underutilized bins where the goal is to see if the bin could be done away with entirely. As mentioned earlier, the total data movement cost of these operations was simply

modeled as the sum of the first dimension of all the relocated items (i.e., size dimension).

Repack requires the choice of a (static) vector packing algorithm, which is then applied to obtain an initial packing of the items; in subsequent *intervals* it again uses a vector packing algorithm to decide how to evict items from overloaded and under-filled bins, and uses it again to decide where to repack them. We tested *Repack* using the off-line version in which all items to be packed are preordered in decreasing order of sum of all dimensions. We measured the quality of the algorithm in terms of number of bins and migration cost. Note that the migration cost is a function of the total size of all the items to be repacked.

For simple packing, previous work had already showed that all the versions of BF and FF have comparable performance in practice. Along with the versions of BF, we also implemented a version of Next-fit (NF). The results on the number of bins were exactly as expected, i.e., BF resulted in far fewer bins than NF. However, we made a curious yet crucial observation. We noticed that NF resulted in considerably less migration cost than BF. (Note that the points corresponding to $k = 0$ and $k = 100$ in Figure 1 correspond to NF and BF respectively.) If the number of bins were the only criterion, then BF and FF and its variants are clear winners. Since the performance of the algorithm is a function of both the number of bins used as well as the migration cost, it meant that the NF algorithm and its variants also had some merit. Thus, BF and NF represented two extremes – NF is an efficient algorithm and produces packings that are not very tight, while BF is less efficient algorithm and focuses on achieving good packings. However, NF has the additional advantage of incurring relatively low migration costs. The observation could be explained by the fact that because NF does not generate tight packings, even when some of the items change their profiles the bins continue to be able to accommodate the items, thus resulting in less migration costs. The take-home message was that *tighter packings will lead to higher migration costs* since the servers are running closer to full capacity and are more prone to having the capacities exceeded by smaller changes in the requirements of individual tasks.

The Hybrid Vector Packing Algorithm: What we needed was a *hybrid* algorithm that could incorporate the best of both the NF and BF algorithms, and we have an excellent candidate. We propose a family of algorithms, which we will refer to as k-bounded BF (on-line) and k-bounded BFD (off-line), for different values of k. We abbreviate these algorithms as kBF and kBFD. NF can be described as an algorithm that looks at *one* bin and decides whether or not to place the next item in it (or to open a new bin). In contrast, BF can be thought of as an algorithm that looks at *all* bins and decides on the best choice of a bin where the item is to be placed (or opens a new bin). The k-bounded BF looks at a fraction k of the bins to decide on the best choice of a bin to place the next item. When $k = 0\%$, the algorithm is same as NF and when $k = 100\%$, i.e., equals the number of bins currently used, the algorithm is the same as BF.

We studied the use of k-bounded BF (kBF) and k-bounded BFD (kBFD) for *Repack* to characterize its effect on the trade-off between the migration cost of repacking at each dynamic interval and the number of bins used. We tested *Repack* with kBF and kBFD for $k = 0\%, 10\%, 20\%, \ldots, 100\%$.

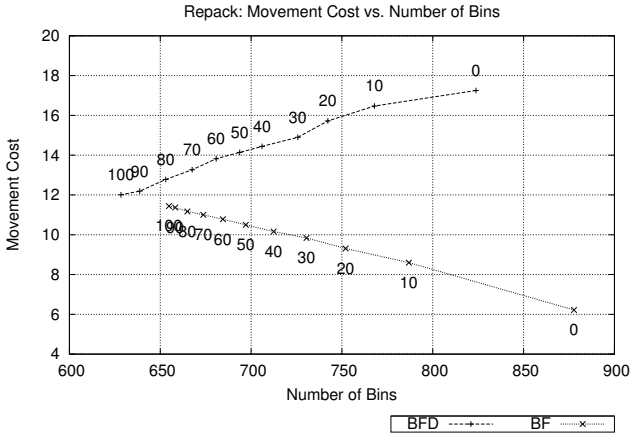


Figure 1: *The effect of varying k on the performance of kBF and kBFD. Note that $k = 0\%$ and $k = 100\%$ correspond to NF and BF respectively.*

Figure 1 highlights the tradeoff between the packing efficiency and the migration cost as k is varied. The first observation is that the tradeoff is non-existent for the off-line version and manifests itself only in the on-line version. In the off-line version, both number of bins and migration cost go down with increase in k. The behavior of the off-line algorithms, for which the input is sorted in decreasing order of some measure of the item, is explained as follows. As k increases, it is no surprise that the number of bins becomes smaller since every item has more choices of bins to be placed. (In fact, the reason for the decrease in the number of bins also applies to the on-line case.)

For the off-line case, at the start when the items are "large", the bins are packed loosely. As the items get smaller, the bins get packed increasingly tightly. However, as k increases, the algorithm packs the last k bins less tightly (since it has more choices) causing lower migration costs from these k bins in the next interval. We have already argued that as bins get packed less tightly, the migration costs start to drop. In the on-line case, since the items appear in random order, all bins packed to roughly the same level of tightness. Therefore, for the on-line case, fewer bins translates very simply to greater migration costs. Our experiments (Fig 7 in Section 7.3 of the Appendix) support the above arguments. Another interesting observation is that the biggest drop in number of bins happens when we go from $k = 0\%$ to $k = 10\%$. This is not surprising since having a few choices is always better than no choices. However, the marginal gain of more choices starts to drop as the number of choices increases.

The real merit of on-line kBF can be seen in the fact that it is possible to "exploit the tradeoff" and optimize the cost of the algorithm by picking the value of k that balances the packing cost and the migration cost, thus providing finer control on the combined total cost. This behavior stays consistent with tests in higher dimensions (see Figure 6 in Section 7.2 of the Appendix).

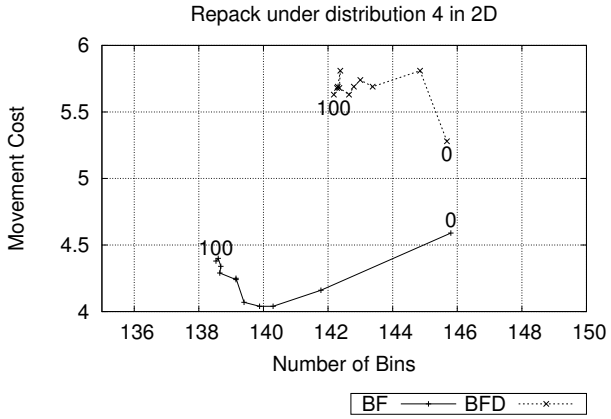
Experiments with Other Distributions: As pointed out by Panigrahy et al. [13], practical resource allocation applications vary widely in terms of how heterogeneous they are in their resource requirements. It is therefore important to perform comprehensive experiments with a variety of distributions under a variety of correlations across dimensions. The following set of experiments were inspired by the work of Panigrahy et al. [13] and Caprara and Toth [4].

The *Repack* algorithm was tested on 200 different simulated data sets each with 1000 items. In each test case, the initial vectors were randomly generated using values from $U(0, 1]$ and distributions from Caprara and Toth [4]. Then for each of 4 *intervals* the values of these vectors were changed in all dimension (except the first dimension, which we refer to as the static size dimension), again generating values from $U(0, 1]$ and other distributions. This is consistent with what is likely to happen in storage systems where working sets do not significantly change in size over a small time *interval* [17]. We use the following 8 interesting distributions from Caprara and Toth [4]. Distributions *Caprara 1* through *6* correspond to $U[0.1, 0.4]$, $U[0, 1]$, $U[0.2, 0.8]$, $U[0.05, 0.2]$, $U[0.025, 0.1]$, and $U[0.133, 0.667]$, respectively. Distribution *Caprara 7* corresponds to $U[0.133, 0.667]$ for odd dimension i and to $U[v_i - 0.067, v_i + 0.067]$ for dimension $i + 1$, where v_i is the value sampled for dimension i. Lastly, *Caprara 8* corresponds to $U[0.133, 0.667]$ in dimension i and $U[0.733 - v_i, 0.867 - v_i]$ in dimension $i + 1$. Thus distributions Caprara 7 (and 8) correspond to a positively (resp., negatively) correlated distributions.

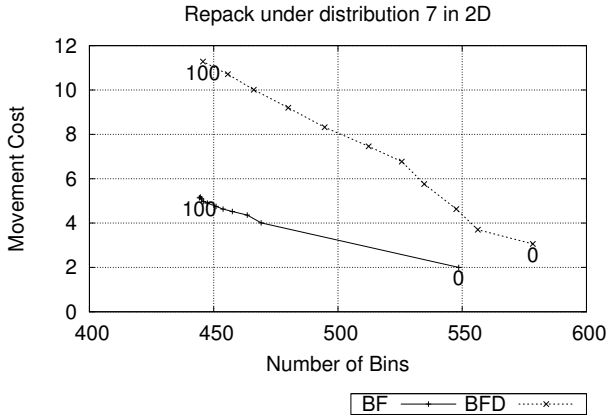
Experiments with different values of k would seem to strongly suggest that 100%-bounded BFD is always the better choice for *Repack* than 0%-bounded BFD. However, the behavior under some of the above distributions suggests otherwise. Figures 2(a) and (b) show the performance of *Repack* with vectors from the distributions *Caprara* 4 and *Caprara* 7. *Caprara* 7 is a correlated distribution and is effectively a 1-dimensional vector packing and its behavior follows a predictable path. *Caprara* 4 has a small variance and other than a big jump when k becomes non-zero, its trajectory stays within a small region. For these and other distributions it is obvious that the best choice of k for *Repack* using kBF or kBFD will depend on the particular cost proportion between number of bins and movement cost. The kBF and kBFD simply offer us greater choice in the family of algorithms that could provide us better performance.

4 Vector Repacking with Replicas

In many resource allocation applications (e.g., storage systems), migration of tasks is often prohibitively expensive. In the previous section we saw how to compensate for it by picking the right value of k in the kBF packing algorithm. Another way to lower or even eliminate data migration costs is to replicate and over-provision the resources at the start [17]. The storage of redundant copies of the items on different servers is a technique already used in storage systems for achieving higher fault tolerance and robustness. (However, here we argue that it also results in lower migration costs.) For example, in Verma et al. [17], the system selects one of the copies of each item to be its *active* copy. Other copies remain on different servers as inactive. When a server exceeds it capacity, then the system may deactivate one or more of the items whose active copy is on that server. For each of



(a)



(b)

Figure 2: *Performance of* Repack *with input vectors from the distributions (a)* Caprara *4 and (b)* Caprara *7*

the deactivated copies, the system then has to choose one of the inactive copies (on a different server) and make it active, thus effectively shifting the load corresponding to that item from one server to another. However, this does not mean that the migration cost can be totally eliminated. When an item switches to a different active copy then synchronization of the copies may be needed since only the active copy is involved in computation and may have undergone some changes. Synchronization is the process of ensuring that the copies are consistent. In the worst case, synchronization costs can be as high as migration costs. However, it is estimated that synchronization costs are on the average considerably lower than migration costs. Thus the replication approach, attempts to tradeoff replication and copy synchronization costs with that of migration costs.

We considered the effect of over-provisioning resources to store replicas of entities with the goal of lowering data migration costs in the event that parameters (such as workload intensity) of entities change. We developed an algorithm called *Replicas* that creates multiple copies (replicas) of each item and strategically places the copies on different servers. When the profiles of the entities change, the algorithm adapts by just selecting a different active copy of the multiple copies without having to move the item to a different location. This switching of active copy will incur some synchronization cost since a fraction of the item may have been dirtied from the time that the copy was made.

4.1 The *Replicas* Algorithm

The *Replicas* algorithm is a modification of the bin packing heuristic to be used in assigning tasks/items to servers. It consists of two parts: REPLICA ALLOCATION (Algorithm 1), and ACTIVE REPLICA SELECTION (Algorithm 2). Given the ordering and packing strategies, and given α, the over-provisioning factor, REPLICA ALLOCATION makes repeated scans of the n items (using the order determined by the given ordering strategy) and places the replicas in $M(1+\alpha)$ bins using the given packing strategy, where M is the number of bins needed by that algorithm to pack one copy of all items. Note that the cost of Algorithm 2 is the cost of data movement involved in placing replicas in bins.

Algorithm 1 Replicas: REPLICA ALLOCATION

Require: V: list of n entities;
α: over-provisioning factor;
p: packing rule; q: ordering rule

`initialize` B to array of n empty bins
`PackOneCopyOfAll(`B`,` V`,` p`,` q`)`
add $\lceil \alpha \cdot M \rceil$ new empty bins to B
while (there is space to store more replicas of any item) **do**
  `PackReplicas(`B`,` V`,` p`,` q`)`
end while

Algorithm ACTIVE REPLICA SELECTION is used to determine which of the multiple replicas of each item is to be the "active" replica. All other replicas are

inactive and are in standby mode for possible later activation. Algorithm ACTIVE REPLICA SELECTION is executed immediately after Algorithm 1 and after any significant changes in item profiles. In storage system applications, this would be done at periodic intervals or after significant changes in load intensities [17]. For a relatively modest cost of storing multiple replicas, the advantage of switching active replicas is that capacity constraints of servers can be achieved with minimal data movement. Note that replicas are placed strategically and the choice of active replicas is made in a manner such that servers with no active replicas can be "turned off" or put in lower energy states to reduce power consumption.

Algorithm 2 Replicas: ACTIVE REPLICA SELECTION

Require: V: list of n entities;
α: over-provisioning factor;
p: packing rule;
q: ordering rule;
B : array of packed bins;

```
let E = ϕ // a set of emergency bins
let O = ϕ // subset of active bins
for all (i ∈ V sorted according to rule q) do
  let C = set of open bins that contain replica of item i
  Pack item i in a bin in C using packing rule p
  if (no such bin) then
    Pack item i in any bin containing it using packing rule p
    if (b is such a bin) then
      O = O ∪ {b} and set bin b as active
    else
      Pack item i in a emergency bin using packing rule p
    end if
  end if
end for
```

Algorithm ACTIVE REPLICA SELECTION maintains a list of currently chosen active replicas. Bins/servers that contain at least one active replica are called "active" bins/servers. Initially there are no active replicas or bins. Items are scanned in the order determined by the ordering rule q. If any of the replicas of that item are in bins that are already active, and if that bin can accomodate the load for that item, then that replica is chosen to be the active one. If none of the replicas are in active bins, then one of the bins containing one the replicas of that item is activated and the corresponding copy of the item in the bin is made the item's active replica. If none of the bins with replicas can handle the load increase caused by making the item active, then an emergency bin is opened and a new replica is placed in that bin and made active (as is the new bin). Note that the cost of Algorithm ACTIVE REPLICA SELECTION is the data movement needed to place items in emergency bins (equal to the total of the sizes of the emergency bins) and the number of emergency bins used.

4.2 Experiments

We tested the *Replicas* algorithm on 200 data sets with over-provisioning factor of 0, 1000 items, each item represented by two-dimensional vectors with randomly generated values from $U[0, 1]$, and with 4 iterations of changes in the second dimension (load). Since switching between copies incurs a synchronization cost, we studied the viability and cost of replication with different dirty ratios. (*Dirty ratio* is the percentage of the item "dirtied" or modified since the last time a replica was made. In other words, it is the percentage difference between the original copy and its replica on another server.) As in *Repack* we tested *Replicas* using FFD-EL2 and BFDSum.

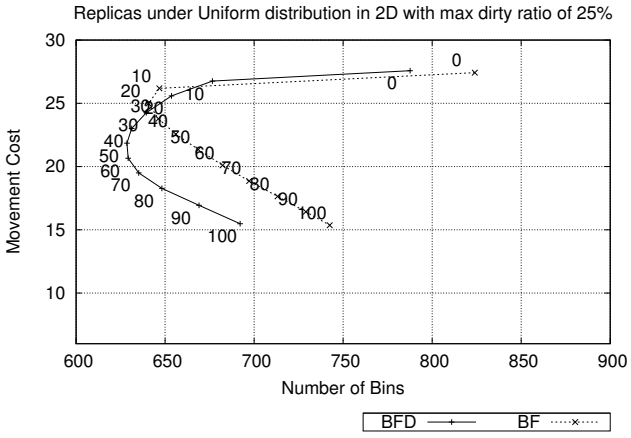
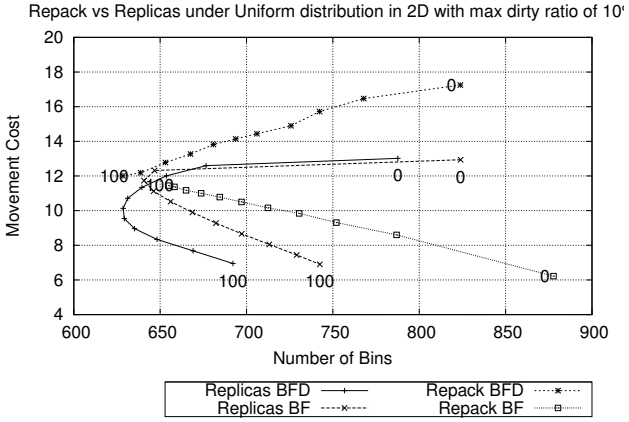


Figure 3: *Performance of Replicas with k-bounded BFD and BF for different values of k. Uniform distribution in 2D. Maximum dirty ratio of 25%*

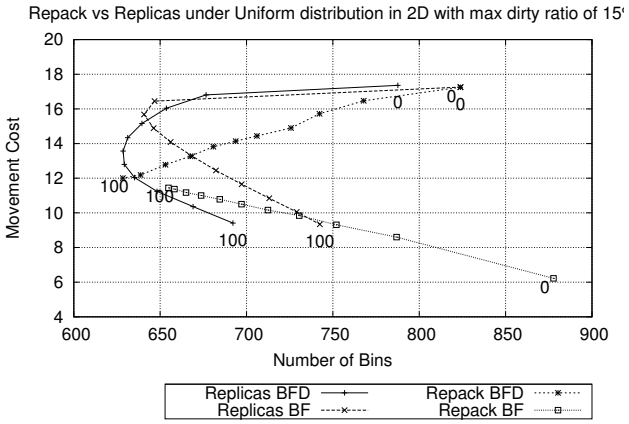
Figure 3 shows that $k \approx 50\%$ for the off-line case ($k \approx 30\%$ for on-line) produces the best performance in terms of lowering the number of bins used. Algorithms BFD and BF produce more efficient packings (than NFD and NF) for the static version of vector packing, but these packings are too tight for the dynamic version, allows less space for replication, and

has less room to adapt to any changes in the profile of the items. Consequently, new emergency bins need to be opened. On the other extreme, NF uses more bins but has more room to adapt to abrupt changes in demands. Thus *Replicas* with k-bounded BF and BFD provides clear trade-offs in terms of migration costs versus the number of bins. The right choice of k will depend on the application and the relative ratios of the migration costs versus number of bins. As with *Repack*, the biggest drop in the number of bins resulted when going from $k = 0\%$ to $k = 10\%$. When k initially increases, the gain in number of bins is tremendous, which in turn means fewer replicas, and smaller gains in migration costs. The drop in migration cost continues with increase in k all the way to $k = 100\%$. However, at some point the gains are nullified by the increase in the number of emergency bins created and hence the C-shaped curve. Higher values of k results in fewer bins for the first copy and therefore less space for the replicas and greater number of emergency bins.

Figure 4 compares the performance of *Replicas* and *Repack*. For lower dirty ratios (Fig. 4(a)), *Replicas* is better than *Repack* since it adapts to change without incurring large migration costs while providing more fault tolerance and using a number of bins that is competitive with *Repack*. When the dirty ratio is high (Fig. 4(b)), the cost of synchronization for *Replicas* becomes a dominant factor. For dirty ratios under 15% over a dynamic interval, *Replicas* performs as well if not better than *Repack* in terms of migration cost.



(a)



(b)

Figure 4: *k-bounded Replicas and Repack. Uniform distribution. a) Maximum dirty ratio of* 10% *b) Maximum dirty ratio of* 15%.

5 Experiments with Real Trace Datasets

We finally tested our suite of algorithms on a storage systems application. Real trace data was generated from accesses to the university storage system. This enabled us to compare *Replicas* with the best existing algorithm called SRCMap to manage replicas in storage systems.

We used the same traces that were used to test the performance of *SRCMap* by Verma et al. [17]. The traces included all I/O data requests over a period of 480 hours to 8 independent data volumes residing on 8 different disks. The traces were used to infer data about the average load intensities of each data volume over each period of one hour length. These values were then used as the realistic input that rep-

resents dynamic workload profiles of the data sets over each interval of one hour. Also for each data volume, the working set of the volume was determined by aggregating all data I/O requests to that volume.

We ran the experiments with 8 data volumes each with load capacity levels of 125, 196, 196, 196, 196, 196, 145, and 145 IOPS respectively. Each data volume set aside 20% percent of its storage capacity as replica space to hold working set copies of other volumes. The data volumes had storage capacities of 270, 7.8, 7.8, 10, 10, 20, 170, and 170 GBs respectively. Each disk $D_i \in [D_n]$ has a working set V_i. Every hour each $V_i \in [V_n]$ is assigned a new average workload value for the length of the interval. Every 24 hours $V_i \in [V_n]$ is assigned a new working set size value for the length of the 24 hours interval. Each working set V_i is replicated by storing its copies on multiple disks from $[D_n]$. Every 24 hours both *Replicas* and *SRCmap* apply their replica allocation algorithm, and every hour they both apply their target mapping algorithm. *Replicas* was run with over-provisioning factor of $\alpha = 0$. Since each working set V_i has its main copy stored in the primary storage section of disk D_i each disk D_i serves all requests to V_i whenever it is active, even if another active disk D_j has a replica of working set V_i. No emergency hosts were set up in *Replicas*. We assumed that in the worse case, all data volumes would be active.

Table 1 compares the performance of *SRCMap* with versions of *Replicas* using 10 different vector packing (VP) strategies. Four of these use BF, four use FF, and two of them use NF replica allocation. Six of the ten flavors of the *Replicas* algorithm were able to service the requests of all working sets using less active hosts on the average than *SRCMap*. The average number of bins using BF and BFD resulted in about 20% decrease in the number of bins. Under this real trace distribution dealing with a small number of servers, k-bounded *Replicas* with $k = 100\%$ was the best option regardless of the cost proportion between movement cost and number of active hosts. The online versions performed better than off-line versions of BF and FF. The FF algorithm performed the best and showed an improvement of over 50% over SRCMap in number of bins.

Table 1: Comparing *Replicas* to *SRCMap*

Algorithm	**Avg # Active Hosts ± Std. Dev**	**Migration Cost Cost**
Replicas NF	3.30 ± 0.63	10.02
Rep 50% BF	3.39 ± 0.66	9.59
Replicas BF	2.22 ± 0.75	9.92
Rep 50% FF	3.07 ± 0.77	9.56
Replicas FF	1.39 ± 0.74	9.95
Replicas NFD	3.29 ± 1.36	11.47
Rep 50% BFD	2.88 ± 1.22	10.59
Replicas BFD	2.36 ± 0.87	10.12
Rep 50% FFD	2.79 ± 1.21	10.55
Replicas FFD	1.44 ± 0.86	10.10
SRCMap	3.05 ± 1.01	12.13

6 Conclusions

This paper considers *dynamic* resource allocation problems, which have high practical relevance in this era of cloud computing where services are provisioned and managed dynamically to minimize cost.

Many important dynamic resource allocation problems can be cast as a vector repacking problem. We have proposed the *Repack* algorithm to address the vector repacking problem; it allows for "incremental" repacking to reduce the cost of migrations due to repacking. In its simplest form, it is based on well-known vector packing algorithms (FF, BF, and NF), and is efficient with regard to the number of bins used and the amount of migration that results. More importantly, we show that a simple variant of the *Repack* algorithm that is based on the kBF (a combination of BF and NF) vector packing algorithm has the right ingredients to be an effective tool for this problem. By picking an appropriate value of k, this proposed variant can be adapted to many different resource allocation applications. The value of k is a function of the relative cost of using an extra bin to that of migrating an item of unit size. We also considered a variant of the vector repacking problem that allows for a limited amount of extra bins to store multiple copies (replicas) of items. Experiments show that a small number of extra bins and multiple copies improves the performance in terms of packing efficiency as well as migration costs. Our experiments show that *Replicas* is a practical tool for resource allocation and power-aware computing for systems that store entities with dynamically (but infrequently) changing characteristics. All our experiments were confirmed with synthetic data sets produced from a collection of different realistic distributions, and with real traces from real systems.

Acknowledgments: This work was partly sup-

ported by NSF Grant CNS 1018262 and a NSF Graduate Research Fellowship DGE-1038321 for MEC. The authors thank Ricardo Koller and Luis Useche for providing the trace data sets used in our experiments.

References

[1] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 217–228, New York, NY, USA, 2010. ACM.

[2] N. Bansal, A. Caprara, and M. Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 697–708, Washington, DC, USA, 2006. IEEE Computer Society.

[3] N. Bansal, A. Lodi, and M. Sviridenko. A tale of two dimensional bin packing. *Annual IEEE Symposium on Foundations of Computer Science*, 0:657–666, 2005.

[4] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Appl. Math.*, 111, August 2001.

[5] S. Y. Chang, H.-C. Hwang, and S. Park. A two-dimensional vector packing model for the efficient use of coil cassettes. *Comput. Oper. Res.*, 32:2051–2058, August 2005.

[6] W. F. de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, pages 349–355, 1981.

[7] Jr. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard problems*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997, 1997.

[8] EPA. EPA Report to Congress on Server and Data Center Energy Efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[9] J. G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3), 2008.

[10] L. T. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM J. Res. Dev.*, 21:443–448, September 1977.

[11] U. Hozle L. A. Barroso. The case for enrgy-proportional computing. *Computer*, 40:33–37, 2007.

[12] K. Maruyama, S. K. Chang, and D. T. Tang. A General Packing Algorithm for Multidimensional Resource Requirements. *International Journal of Computer and Information Sciences*, 6(2):131–149, 1977.

[13] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing. Technical report, Microsoft Research, 2011.

[14] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*, 70(9):962–974, 2010.

[15] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: a power-proportional, distributed storage system. Technical report, Microsoft Research, 2009.

[16] P. Shenoy V. Sundaram, T. Wood. Efficient data migration in self-managing storage systems. In *Proceedings of ICAC 06*, pages 297–300, 2006.

[17] A. Verma, R. Koller, L. Useche, and R. Rangaswami. Srcmap: energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[18] G. J. Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Inf. Process. Lett.*, 64:293–297, December 1997.

7 Appendix

7.1 Experiments with *Repack* on a variety of input distributions

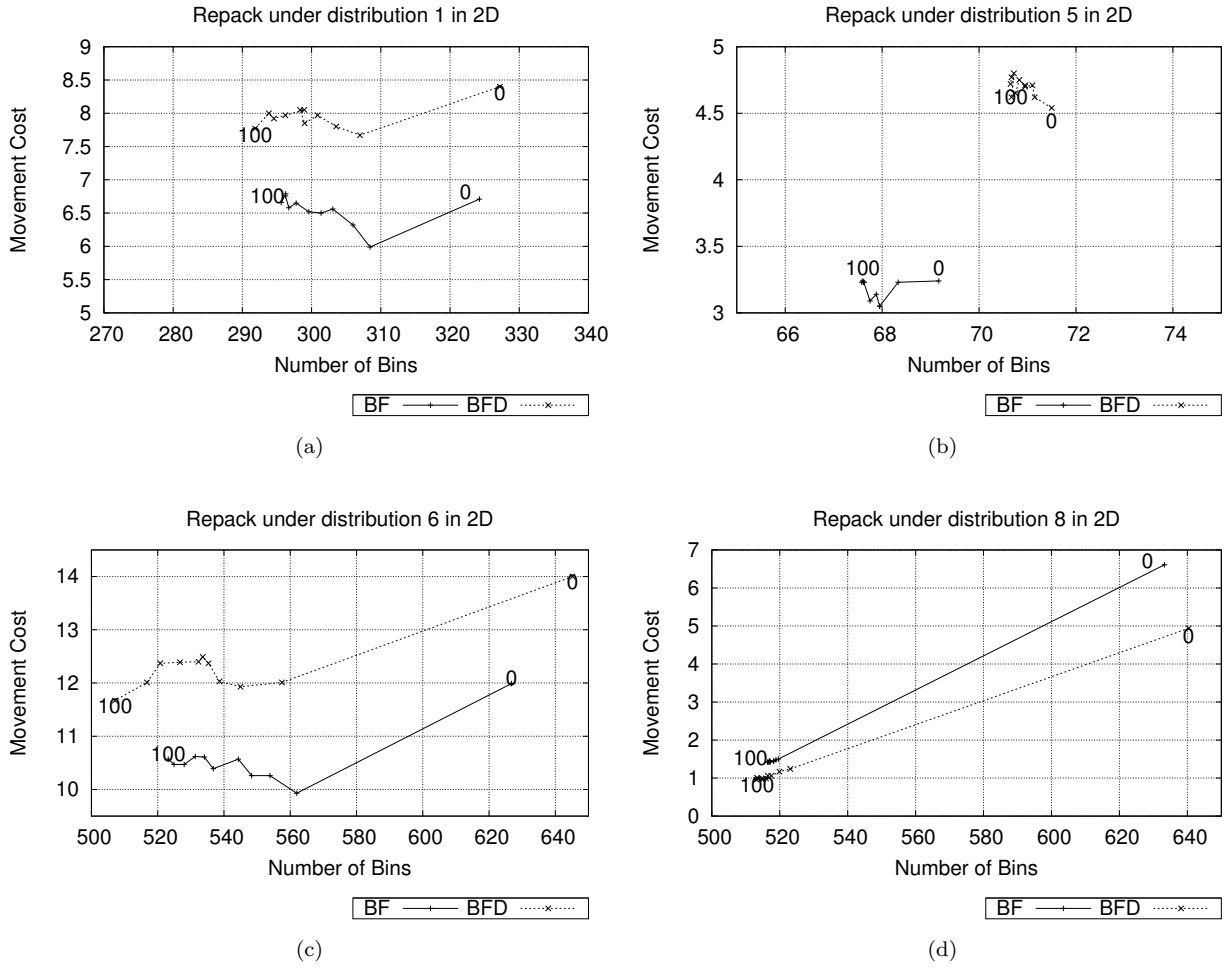
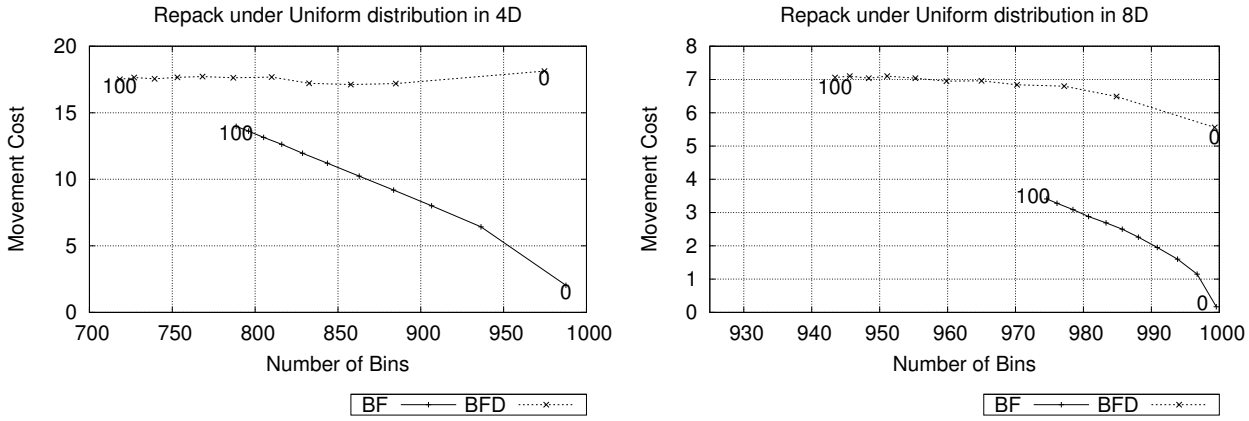


Figure 5: *Performance of* Repack *using kBF and kBFD on four distributions: (a) Caprara 1, (b) Caprara 5, (c) Caprara 6, and (d) Caprara 8*

7.2 Experiments with *Repack* for higher dimensions



(a) (b)

Figure 6: *Experiments with* Repack *for 4 and 8 dimensional inputs*

7.3 Experiments measuring level of bins after first dynamic interval for *Repack* kBF and kBFD

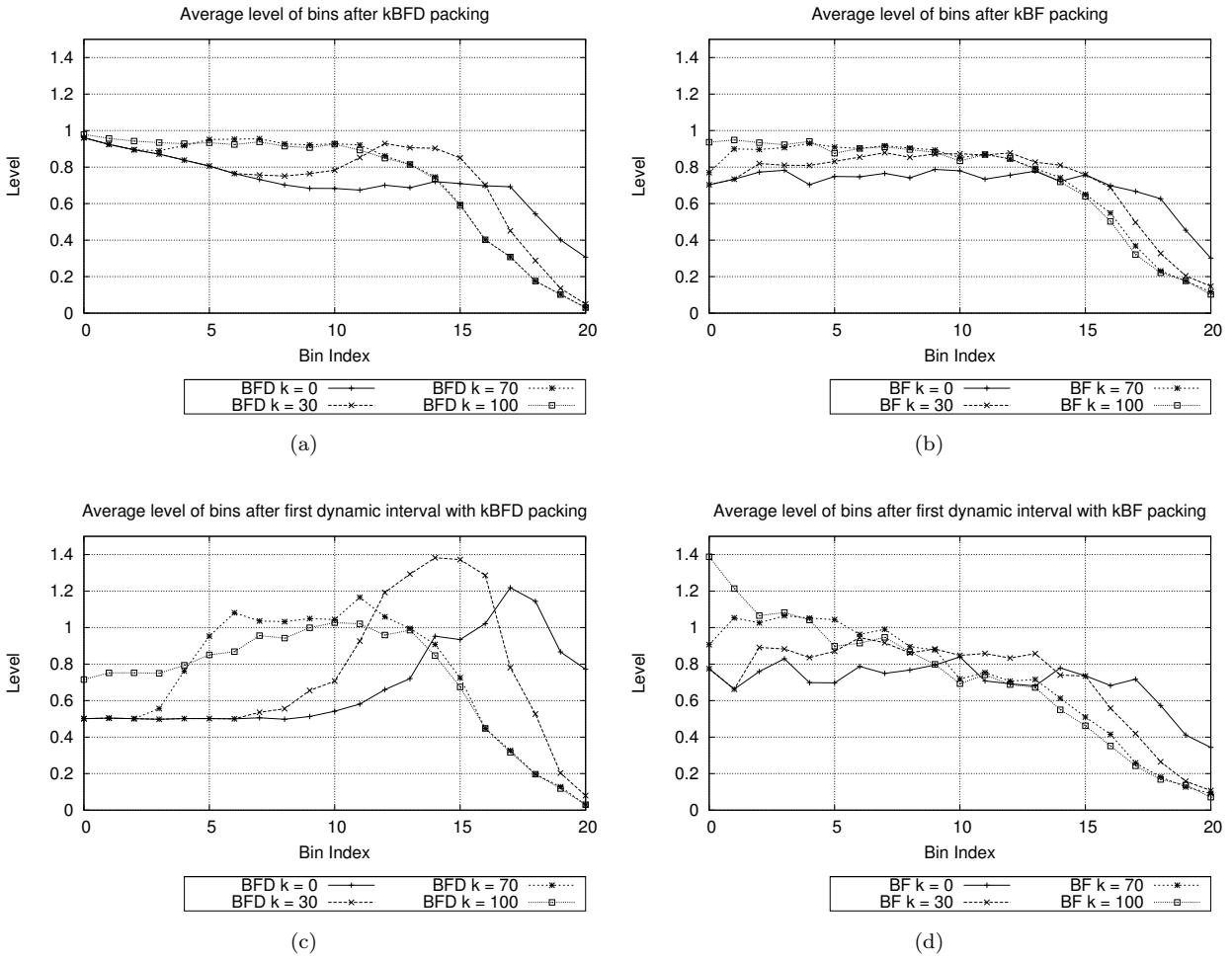


Figure 7: *Figures on the top row show average levels to which bins are filled after the initial packing with* Repack *(a) kBF and (b) kBFD. Figures on the bottom row show average levels to which bins are filled with* Repack *(a) kBF and (b) kBFD after the first dynamic interval when their profiles are randomly changed. As seen in the graphs on the bottom row, several bins have their capacities exceeded and will require migrations of some of the items.*