CS787 Project: Bin Packing: A Survey and its Applications to Job Assignment and Machine Allocation

Brian Kroth

2013-04-19

Abstract

In this paper we present a survey of some common online heuristics to the single dimensional bin packing problem, and their analysis, including some offline variants, especially as it pertains to resource allocation. Through the lens of a case study of virtual hosting at a local university campus, we examine its applicability to the sizing of resources based on task demands and consider some multi dimensional variants as well as some takes on the problem from the perspective of some commercial applications.

1 Introduction

The single dimensional bin packing problem (BPP) can be described as finding a way to assign a list of n items L, each with varying sizes or weights s_i, into the smallest number of identical bins, each of capacity B. Usually, the item sizes are normalized to the range $[0, 1]$ with unit bin capacities. Given as an (integer) linear program, the problem can be stated as follows:

$$\text{Minimize: } \sum_{j=1}^{n} y_j$$

$$\text{Subject to: } \sum_{i=1}^{n} s_i x_{ij} \leq B y_j, \qquad \forall\, j \in \{\, 1, ..., n \,\}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall\, i \in \{\, 1, ..., n \,\}$$

Where the y_j and x_{ij} are each $\{\, 0, 1 \,\}$ indicator variables denoting whether we need to use a bin j (up to n since we need at most n bins in any solution) and whether item i is included in bin j, respectively. This can be extended to multidimensional problem descriptions in different ways as well as variable sized bins, though even in the single dimensional case it is an NP hard problem. Yet, the problem has basic applications to industry in the context of packing real physical objects in boxes [12], resources assignment such as VMs to physical machines [32] [25] [38], and task assignment such as in the context of multiprocessor systems [6], and has such has generated continued interest over the years. Additionally, it has served as a test bed for various advancements in the techniques used in competitive analysis [2]. It is also closely related to the cutting stock and partitioning problems.

In this paper we focus on a survey of several common heuristics for greedy approaches to BPP such as First Fit (FF), Best Fit (BF), Harmonic, and Sum of Squares (SS). We also conduct a brief comparative study of certain aspects of them on our case study's data such as the repacking displacement effect for items that may change in size, and load balance, which is typically consider to be a separate problem. We conclude with an overview of bin packing's uses in some related work and describe future possible areas of exploration.

2 Bin Packing Analysis Overview and Bounds

The problem of bin packing has long been studied to to its many applications and NP hardness, which has been known since at least [21]. In general, the techniques used in solving the one-dimensional version can be divided into online versions, where the algorithm does not have foreknowledge of the future input stream, and offline versions where the algorithm can examine the entire set of input data, bounded and unbounded space, depending upon how many bins are kept "open" for consideration at a time, and greedy approximations and integer (IP) and linear programming (LP) approaches.

Online analyses have traditionally focused on the worst case competitive ratio α of an algorithm A to that of an optimal OPT, sometimes defined as $A(L) \leq \alpha OPT(L) + c$ for some constant c, though

other texts prefer a lim sup based definition. This is usually obtained by means of a work function to attribute some cost to an item's bin assignment [4] or by constructing a worst case list where $OPT(L)$ can still be reasonably calculated and then carefully accounting for the possible combinatorial arrangements given by the algorithm's solution. However, more recently work has been concerned with determining average case analyses of the various heuristics under certain item distributions by looking at the expected waste in a given bin configuration compared with that of OPT, which is sometimes approximated via novel pseudo-polynomial time LP calculation techniques [14], which is the best we can hope for in this case.

Of the approaches outlined above, we will focus primarily on the greedy algorithms, which are also the ones more commonly used in practice. According to [32], IP or LP approaches are 1) relatively too expensive to compute except for small values of n (eg: 10s or 100s), and 2) in the case of the more common integer instances of the problem, require non-obvious rounding techniques which may result in worse approximations.

However, it is worth noting that it is due to the LP methods used by Courcoubetis and Weber (CW) in [12] [10] [11] that we have very useful general bounds for average case analysis of different item distributions. By mapping the solutions for any discrete distribution F onto the convex cone, they were able to show that there exists an optimal algorithm OPT for the three possible cases corresponding to to whether the solutions were inside, on, or outside the cone, and that the expected waste of each of these three cases was $O(1)$, $\Theta(\sqrt{n})$, $\Theta(n)$, correspondingly. [14]

Combined with the perfect packing theorems of Coffman et al. in [7] and others, which says that for certain integer valued lists whose sum of the equally occurring items is a multiple of the bin size there exists a perfect packing arrangement (ie: with no wasted space), these oft cited papers have been used, for example, to derive some average case analyses under certain discrete, usually uniform, distributions for several greedy algorithm heuristics such as SS, FF, etc. [15] [29] in cases that would otherwise be intractable to get a reasonable bound on OPT for comparison to A.

There are several other general and useful bounds shown by Yao [39] in what Borodin called the first use of competitive analysis [4] for his work on Refined First Fit (RFF), a variant of the Next Fit algorithm, itself a simplified version of First Fit that only keeps a single bin open at any time.

For instance, he shows that any online algorithm, regardless of computational cost, can do no better than a competitive ratio of $\frac{3}{2}$. A bound which was later improved to 1.536... by Liang [27]. The argument consists of constructing a list $L = L_{\frac{1}{6}}L_{\frac{1}{3}}L_{\frac{1}{2}}$ of sublists of items of sizes $\frac{1}{6}-2\epsilon$, $\frac{1}{3}-\epsilon$, $\frac{1}{2}-\epsilon$, respectively, for an $\epsilon \in (0, .01)$ so that it should be clear to see that when $|L_{\frac{1}{6}}| = |L_{\frac{1}{3}}| = |L_{\frac{1}{2}}| = n$, then $OPT(L_{\frac{1}{6}}) = \frac{1}{6}n$, $OPT(L_{\frac{1}{6}}L_{\frac{1}{3}}) = \frac{1}{2}n$, and $OPT(L) = n$. However, intuitively, if an online algorithm A optimally packs the first list so that $A(L_{\frac{1}{6}}) = \frac{1}{6}n$, then it doesn't have enough space later to optimally pack larger items as well. Similarly, if it saves space up front to pack the larger items optimally, then it necessarily cannot pack the smaller items optimally. So, by a careful accounting of the possible configurations, Yao is able to show that $\max\{\frac{A(L_{\frac{1}{6}})}{OPT(L_{\frac{1}{6}})}, \frac{A(L_{\frac{1}{6}}L_{\frac{1}{3}})}{OPT(L_{\frac{1}{6}}L_{\frac{1}{3}})}, \frac{A(L)}{OPT(L)}\} \geq \frac{3}{2}$.

Also in this same paper, using similar ϵ construction of lists, Yao derives a simple lower bound of d on the competitive ratio of any d dimensional online (vector) bin packing algorithm, which to our knowledge, has not been improved upon.

Finally, we note that while online greedy approximation approaches have been our focus, there are some results [16] showing that for any ϵ there exists an offline asymptotic polynomial approximation scheme A_ϵ so that the competitive ratio of A_ϵ is at most $1+\epsilon$, though with large constants so that it is not a reasonable competitor to something like MFFD [29]. One improvement upon that result was the differencing method of Karmarkar and Karp (KK) [22] which showed how to achieve $KK(L) \leq OPT(L) + O(\log^2 OPT(L))$ using complex LP and ellipsoid rounding methods, but at the cost of $O(n^8 \log^3 n)$ [14], thus it has usually be considered too costly and complex to be generally used in practice.

In the following two sections, we continue with a survey of some First-Fit and Best-Fit based heuristic methods.

3 First Fit Based Heuristics

In this section we briefly discuss some greedy heuristic algorithms related to the First-Fit (FF) algorithm.

FF is an $O(n \log n)$ [20] unbounded space online algorithm which uses the rule of, for a fixed order of bins, simply placing the arriving item into the first "feasible" bin in the ordering, where by feasible we simply mean one with a sufficient free space gap to fit

it. If there is no feasible bin available, then FF opens a new one and places the item into it. Repacking is not generally allowed or considered in most BPP formulations.

FF also has an offline version called First-Fit-Decreasing (FFD), also $O(n \log n)$, in which the entire list is first sorted in decreasing order of item sizes so that large items are processed first, filling in the largest gaps possible as the list is worked through in order to achieve an approximation ratio of $\frac{11}{9}$. [21]

An even simpler heuristic, which can be implemented in $O(n)$ time, can be seen in Next-Fit (NF) which keeps only one open bin under consideration. If the arriving item won't fit in that bin, then that bin is closed for future consideration and the item is placed in a new bin. NF achieves a competitive ratio of 2 since lists such as $n = 2k$ copies of $\{ \frac{1}{2}, \frac{1}{2k} \}$ will cause it to place each sublist into its own bin with $\frac{1}{2} - \frac{1}{2k}$ space wasted, leading to a total of $2k$ bins, where as OPT would place a pair of the $2k$ $\frac{1}{2}$ pieces together in their own bin and the $2k$ $\frac{1}{2k}$ pieces together leading to $k + 1$ bins in total.

On the other hand, in [21] Johnson et al. showed that FF has a competitive ratio of $\frac{17}{10}$ using a similar worst case list construction that we previously cited from Yao [39] - $L = L_{\frac{1}{6}} L_{\frac{1}{3}} L_{\frac{1}{2}}$ - so that it should be easy to see that $OPT(L) = n$. In a simplified version of the proof due to Borodin [4], it then sets up a work (weight) function split by item size intervals that attributes so much of each bin to every item. If we change our perspective then and view the weight of each item as the amount of bin space it uses up, including wasted space, then it can be shown by enumerating the possible combinations in each bin, that the work (weight) of an optimally packed bin is at most $\frac{17}{10}$.

In [39], Yao extends the basic idea of grouping items together in this fashion into a new algorithm Refined-First-Fit (RFF) by classifying items by more disjoint ranges: $(\frac{1}{2}, 1]$, $(\frac{1}{3}, \frac{2}{5}]$, $(\frac{1}{3}, \frac{2}{5}]$, $(0, \frac{1}{3}]$. Items in each range are packed together using a NF heuristic within each class, except that certain smaller items are packet into bins with larger items, in order to make use of the otherwise wasted gap in those bins. Through careful accounting for the possible combinations Yao is able to show that RFF achieves a competitive ratio of at most $\frac{5}{3}$. One interesting feature of RFF is that it only needs to keep one open bin for each class. Thus, unlike Best-Fit or First-Fit, it has bounded space requirements.

This idea of classifying items into a fixed number of ranges naturally extends to the $HARMONIC_M$ algorithm proposed by Lee in [24]. In it they define ranges $I_k = (\frac{1}{k+1}, \frac{1}{k}]$ for $k \in (1, M]$ and $(0, \frac{1}{M}]$ for I_M, so that exactly k items will fit into I_k when it is full, so that like RFF, $HARMONIC_M$ only needs to keep a single bin for each class open at a time. Further, evaluations of HARM show that there is little improvement on the competitive ratio of 1.6910... for $M > 12$, so $HARMONIC_{12}$ can be viewed as a bounded space online algorithm. Note also that the number of bins in the solution to an input L that $HARMONIC_M$ produces is invariant to the order in which the items arrive, since they all go into a fixed bin classification. Finally, like RFF, Refined-HARMONIC extends harmonic by packing smaller items into I_1 bins which may have just less than $\frac{1}{2}$ of their space "wasted" in order to a achieve a worse case competitive ratio of $1.6359 < \frac{5}{3}$ of RFF.

4 Best Fit Related Heuristics

In contrast to the FF variants, the recent Sum-of-Squares (SS) algorithm takes its inspiration from the Best-Fit (FF) algorithm. Like FF, BF is another very common, simple, and effective greedy heuristic which keeps an ordered list of bins and places an incoming item into the feasible bin that will result in the least free space gap remaining in that bin, though in this case the ordered list of bins only affects the algorithm's efficient ($O(n \log n)$) implementation, not its correctness. The online BF has been shown [21] to have the same competitive ratio of $\frac{17}{10}$ as FF, and the natural offline version, Best-Fit Decreasing (BFD), also obtains FFD's worst case approximation ratio of $\frac{3}{2}$.

However, work by Coffman et al. [8] [9] and Shor [35], showed that BF does better than FF in the average case, by generating expected waste of $\Theta(\sqrt{n} \log^{\frac{3}{4}} n)$ compared with $\Theta(n^{\frac{3}{4}})$ under a continuous uniform distribution on $(0, 1]$, and $O(\sqrt{n} \log k)$ versus $\Theta(\sqrt{nk})$ when a discrete uniform distribution of $U\{k, k\}$ is used, where $U\{j, k\}$ represents the discrete uniform distribution of the set $\{ \frac{1}{k}, ..., \frac{j}{k} \}$ for $j \in [1, k]$. [1] Coffman et al. further show that when $k \geq j(j+3)/2$, then BF is "stable". That is, the expected waste is bounded by a constant regardless of the size of the input list L. This corresponds to the first case of the aforementioned CW results.

Interestingly, the discrete average case results for BF are derived using a Markov Chain model by first recognizing that any packing configuration can be represented by the residual space left in the bins, but the order itself doesn't matter. Then, the possible arriv-

ing items represent the transitions from a possible bin configuration to another.

In [1] Mitzenmacher and Albers extended this idea to a Random-Fit (RF) heuristic. Unfortunately, the competitive ratio RF achieves is no better than NF's 2, but by extending it to RF(2+) with a Power of Two Choices technique so that it examines 2 (or more) possible state transitions (bins) for every incoming item, they are able to show that, for lists of sufficient size, it achieves a competitive ratio of $\frac{17}{10}$ just as BF and FF do.

The Sum-of-Squares (SS) algorithm grew out of this work in analyzing the average case of algorithms like BF under certain discrete distributions like $U\{j,k\}$ that was described above, and as such only handles cases of integer sized items and bins, or those that can be scaled to be, so often the normalization factors are dropped when discussing it.

In the case of BF, it was noted that it does particularly well in the case of symmetric distributions - those where $\mathbb{P}[s_i = s] = \mathbb{P}[s_i = B - s]$, so that there is a high likelihood of perfectly packing a new item into a bin since there probably already exists a partially filled bin with a free space gap of that size, so there is close to no waste.

The SS algorithm attempts to extend this idea to non-symmetric distributions by reacting to the input stream to maintain a relatively consistent gap size amongst all its open bins. This is done by placing a new item into the bin that rather than just minimizing the sum of the gaps in the remaining open bins, minimizes the square of the gaps, since "for a fixed sum of variables, their sum of squares is minimized when they area as close to equal as possible". [15] Stated mathematically, the objective function is:

$$\text{Minimize: } \sum_{1 \le g \le B-1} N(g)^2$$

Where $N(g)$ represents the number of currently open bins that have gap of g.

In [15] Johnson et al. note that for an incoming item of size s there are three cases:

1. The item starts a new bin, so that the sum of squares increases by $2N(B-s)+1$,
2. It perfectly fills an old bin, so that the sum of squares decreases by $2N(s)-1$, or
3. It goes into a gap of size g which minimizes $N(g)-N(g-s)$ so that the sum of squares decreases by $2(N(g)-N(g-s))-2$

so that no squares need actually be computed.

However, they do not find a way to appreciably decrease the case 3 operation to less than a search over all bin gap sizes, so SS results in $O(nB)$ time rather than $O(n \log B)$ that BF takes, though both require $O(n)$ space since they do not keep a fixed number of bins like NF, RFF, HARMONIC.

Further, although it is shown to have a worst case competitive ratio of 3 for general distributions, in particular those classified by CW as $\Theta(n)$ expected waste, so that it does not violate Yao's result [39], in the case of the so called "perfectly packable" and "bounded waste" distributions, SS obtains ratios close to 1 and at most 1.5 in the case of $O(\sqrt{n})$ expected waste distribution. [14]

Finally, we note that for asymptotic input list sizes, a variant of SS, is able to "learn" the discrete input distributions that may vary over time by are still confined to a fixed set in order to achieve $O(1)$ expected waste by selecting a new randomized SS variant that is tuned to the particular distribution [14]. However, even the authors consider this approach to be too impractical.

5 Case Study

The Computer Aided Engineering center at the College of Engineering of UW-Madison has a fairly complex virtual hosting infrastructure that it provides for the college and other university wide services. The ultimate issue we would like to be able to answer is how best to size our server infrastructure to suit the service's needs using some variant of bin packing, instead of trial and error. In order to explore how we might do this, we first describe the system as it is currently laid out as follows.

5.1 Background

There are roughly 800 individual production virtual hosts (vhosts) of various types served by roughly 25 backend virtual machine (VM) servers that are proxied by 4 frontend servers. Each vhost has certain properties associated with it in a database such as an editing group, the certificate domain it belongs in [1], and a type, as well as a number of others. In order to save on memory footprint required, but still

[1] We use wildcard certificates as much as possible to save on IPv4 address requirements on the frontends, though we make use of IPv6 addresses extensively as well.

retain certain security features such as running different websites under different users, the vhosts are combined into roughly 650 security domains by these three features, and a single standalone Apache web server instance is dedicated to running each of these, with an individual `VirtualHost` directive for each vhost that belongs to that security domain.

A vhost's type might be one of HTML, Mod PHP, Mod Perl, etc. Each backend VM implements one of these types and represents a standard hardware configuration (eg: 4 GB RAM, 2 vCPU, etc.) for that type. The type of a vhost determines which type of VM a vhost can be assigned to and controls some of the vhost's default configuration parameters including how much memory a barebones vhost of that type is expected to take up. We are generally more concerned with memory than CPU usage, though certainly each request occupies a certain amount of CPU time. However, individual vhost code and settings customizations can increase those requirements, and since it is not currently practical to load test every individual vhost accurately, currently we tend to fairly crudely estimate the size of a given vhost type by using what we expect to be typical values and then possibly adjusting it by a factor based upon the load its requests generate on the system.

5.2 Vhost Workers

Currently a rudimentary assignment routine is used to balance the number of the vhosts across the available number of backends for a given type by considering a `workers` property on each vhost, which is currently set manually, while avoiding moving the "heaviest" vhosts if possible to prevent service interruption. The `workers` property is an integer field which is (ab)used in the following ways:

1. It represents the number of spare/idle Apache child processes [2] we configure for the vhost's security domain [3] to keep around to process incoming requests. The tradeoff here is that since each worker is a full separate process keeping more spares around requires more memory, but lowers the expected response time latency since we need to also incur the cost of a fork.
2. The `workers` property is assumed to represent the relative importance of the vhost (ie: if we expect more requests, then more people must care about the resource it represents), so we configure our monitor systems to check it more frequently. This has the side effect of possibly keeping more children active after they otherwise would have been reclaimed in the case that the vhost goes idle for a period of time.

Currently the `workers` property defaults to a value of 2 for all vhosts in order to try and keep memory requirements low, since most vhosts are not particularly active. If the vhost has more activity than that, then extra children are automatically spawned to handle the incoming requests, and are automatically reclaimed when they've been idle for a sufficient period of time.

On the other hand, certain vhosts get enough activity or have extra redundancy requirements that they are assigned multiple backends and the frontend proxy is configured to balance and failover among them. This is currently controlled through a `secondary backend instances` integer field, and is also controlled manually.

5.3 Vhost Size

If we had

1. a good value for the `workers` property for a vhost that accurately represents the expected number of requests we'll need to process in a given time window, and therefore the number of Apache child processes that will be running at a given time, and
2. a good estimate on the amount of memory required by each of those Apache child processes,

then, we should be able to obtain a reasonable estimate on the memory "size" of a vhost by simple multiplication. [4] If we additionally had an estimate on the average amount of time or CPU resources it took to process a request, we could also obtain an estimate on the CPU "size" of a vhost.

To address item 1 we have analyzed the Apache Prefork and Worker MPM source code and written a simulator to determine the number of active worker processes/threads at each second throughout the day [5]

[2] For most vhost types we use a Prefork Apache MPM rather than a threaded one, so each worker is a separate full process since most user code, especially PHP, doesn't properly support threading.

[3] Currently the maximum value of all vhosts in a security domain is used for that Apache instance, though under the proposed scheme that might be reworked to be the sum of the security domain's vhosts' `workers` property

[4] At least, that mechanism works for Prefork Apaches. For threaded Apaches we also need to divide by a factor of 25 first, since in that case a new full process is not forked until all of its threads have been claimed.

[5] This is the highest granularity the logs will give us.

by replaying access logs. This time series data is then summarized on a per day basis into a table that includes for each vhost and each day min, max, median, mean, and variance of the number of active workers for that vhost throughout that day, so that we can very roughly reconstruct the distribution later should we need to, without needing to rerun our simulator over all of the log data. For this initial experiment we decided to use the average number of workers per day, however for the future we are considering using an exponentially weighted moving average over each day's average or maximum in order to get somewhat smooth reactions to changes in long term vhost activity. One area of future study could be to analyze how best to tune the exponential factor parameter, or if another distribution would provide a better estimate altogether.

To address item 2 for each vhost accurately, we would really need to perform load testing on each vhost independently. Since this is not practical, we decided to obtain an upper bound by loading a vhost with as many extensions and plugins as possible (ie: turn all the available settings on) and load test it to get a reasonable guess at how much memory a single worker could possibly use. We then applied techniques outlined in Section 10.1.1 of [3] in order to estimate the real memory usage of each Apache process, since there's some shared between different worker children. This is a somewhat more difficult task for the threaded Worker MPM type vhosts, so at the moment our results are confined to just Prefork MPM type vhosts. It should be noted that the technique above also doesn't currently consider memory required by threads of other cooperating processes on the system such as `shibd`.

To extend this to a multidimensional vector packing problem, for vhost request CPU resource estimates we can again use log analysis of the average request response times. However, this number unfortunately encompasses extra overhead such as external I/O calls during which a worker may be preempted so it does not accurately reflect actual CPU resource demands. Hence, for the moment we restrict ourselves to only considering the memory dimension.

5.4 Vhost to VM Assignment and Infrastructure Sizing

Thus, for each separate type of vhost, we have a bin packing problem where the capacity of the VM (bin) for that type is fixed, and the vhost sizes (items) vary in size. Our goal then is to determine the minimum number of VMs we need in order to serve all of our vhosts.

First we note that a vhost with a "size" larger than the capacity (ie: RAM, CPU) of a single backend of that type, or perhaps some fraction of it [6], would be a good indication of when we need to split that vhost across multiple secondary backends. This should then be equivalent, for instance, to replacing a ball of size n with 2 of size $\frac{n}{2}$, and then running our bin packing algorithm.

Next, although load balance is perhaps a separate problem, we would also like to consider a bin packing solution's affect on it as follows.

If were also to use bin packing to determine where to actually place each vhost, then we would like our solution to be done in a "stable" and well balanced way so that we do not disrupt service too much by unnecessarily moving vhosts to different VMs even when their "sizes" might change slightly overtime and so that the heaviest vhosts are not generally pitted together as that might impact our ability to handle a flood of requests to a particular vhost gracefully or the loss of a VM without an outage on too many "important" vhosts at once.

Note that in this context our definition of "stable" differs from the that typically used in bin packing contexts. There the term is used to refer to distributions that have bounded expected waste, whereas we mean something closer to the notion of "minimal resizing displacement". For instance, if we were to use FFD to place our vhosts on VMs and one of the larger (and therefore first placed) items was reduced in size (eg: due to a shift in website content popularity, or time of year relative to the academic calendar), then potentially many of the vhosts following it are shifted from their previous VM assignments.

Finally, although rebalancing is usually done off peak request hours, adding new vhosts can happen at any time during the day and they are expected to be provisioned and made available within a very short period of time. Therefore, we consider running our bin packing in to different phases:

1. An online version that takes the current packing configuration and determines the best place for an incoming item of that type.
2. An offline version that runs off hours to rebalance everything.

Note, in case 1 there are at least three things to note:

[6] For these experiments we tried $\frac{1}{4}$.

1. We don't know how large a virtual host will be at first.

2. Our system could actually lock the database the configuration is derived from for a short period and perform an offline FFD algorithm in order to find a better approximation as to where to place a vhost.

3. Some sort of periodic rebalance will still be necessary since vhosts change in size over time and may in fact be decommissioned and leave the system entirely.

5.5 Evaluation

In order to evaluate the FF(D), BF(D), and SS(D) heuristics on our vhost input data we consider the following metrics: [7]

1. Total number of bins required, especially when compared with the current number of VMs in our system.

2. Total waste in the proposed solution.

3. Average waste per bin, and the standard deviation of this over all bins.

4. Average item size per bin, and the standard deviation of this over all bins, as a measure of load balance.

For each heuristic's initial solution, in order to gain some measure of the displacement effect, we also conduct some experiments:

1. Increase and decrease the heaviest vhost by a factor of $\frac{1}{10}$ to simulate a slight drop or slight increase in activity.

2. Increase the lightest vhost by a factor of 2 and 5 to simulate a slight increase in activity.

We then measure the displacement effect as the number of vhosts that change their VM assignments as a result of the changes listed above.

Finally, as an attempt to incorporate some load balancing into the algorithm, we propose a simple extension to the BF(D) algorithm, which we call BF(D)_LB that adds the rule that for each bin, no more than one vhost of size $\frac{1}{4}$ of the VM's capacity shall be allowed in a valid configuration.

[7] Note, we excluded HARMONIC and RF due to time constraints and because they clearly weren't going to satisfy our stability and load balance constraints.

Later we also added the rule to not place more than one copy of a split vhost (due to secondary instances, which is in turn due to over all size relative to a fraction of the bin size) on the same VM. We did this after we noticed that despite preprocessing large vhosts to split them as described above the original algorithms still packed those vhosts together, which largely defeats the point.

We also determined through some trial and error that using almost any form of mean of daily means (rather than max, median, etc.) resulted in the closest approximation of results to what we are currently running in production. At the time of this writing only three months of historical logs had been simulated to obtain daily worker summaries, which is fairly closely aligned with a single semester of activity (our busy period), so it's possible that as we gain more and more historical data different ways of processing that data become more useful. For the results posted here, we simply used an overall average of all available daily averages for each vhost in order to obtain our worker count estimates. They were also verified against several weeks of monitoring report data for a subset of the vhosts so that we believe they are accurate.

For our measured worker process in use memory sizes, we used the maximum value seen by examining the systems over several days. However, for each type, the maximum varies significantly from the mean. This is another area for further refinement.

Two separate vhosts types were considered.

1. `php5` is generally available for all users and contains many small vhosts, with a few medium sized ones.

2. `moodle` is a specialized fork of the `php5` type. It's backends get more resources and has far fewer, but generally much larger vhosts, as well as a few small ones for development and testing purposes.

It should also be noted that in both cases, the distribution of our item sizes is far from uniform (there are many more smaller items), so the Uj, k and many other average case bounds derived for the heuristics do not apply. Rather, unfortunately, determining which of the three cases our distribution is in is NP hard.

5.6 Results

The results for `moodle` type vhosts is given in 5.6. Although, as we have said, there are a couple of very large vhosts in this class that are split across multiple

Table 1: `moodle` results

`moodle`	FF(D)	BF(D)	SS(D)	BFD_LB
Total Bins	1	1	1	4
Difference from Prod.	-4	-4	-4	-1
Avg. Item Size/Bin Mean	561.75	561.75	561.75	888.30
Avg. Item Size/Bin Stddev.	0	0	0	377.07
Total Waste	1394	1394	1394	19058
Waste/Bin Mean	1394	1394	1394	4764.50
Waste/Bin Stddev.	0	0	0	30.31
Largest Vhost 1.1x Increase Displacement	0	0	0	0
Largest Vhost 0.9x Decrease Displacement	0	0	0	0
Smallest Vhost 2x Increase Displacement	0	0	0	0
Smallest Vhost 5x Increase Displacement	0	0	0	0

backend VMs, the standard algorithms all repack the split vhost items back into the same bin. As such, the online and offline versions, actually behave identically, so they are combined in this table. This is contrary to our experience with these vhosts, so we believe that the worker process size estimates must be in accurate, though it's also possible that these vhosts, unlike what we'd assumed earlier for the others, are actually bottlenecked by CPU, so measuring these vhosts' size by memory alone may be insufficient. Although the numbers get a little bit closer to what we see in practice for this case when we use the maximum observed worker process size, then the values for the `php5` data set is blown out of proportion.

The `php5` type vhosts produce slightly more interesting results. For solutions that have more than one bin, we would expect the one that has the least standard deviation of item sizes among the bins to give the most balanced solution. It makes some sense that the online versions of these heuristics should give those since they get a slightly more random ordering of the input. [8] However, it's interesting to note that these layouts also give the most displacement when vhost activity and sizes change.

Though it seems natural that our extra load balance conditions would lead to more bins and extra wasted space, we were surprised that the BFD_LB heuristic didn't actually help the load balance much. This can be seen in both the average item size deviation as well as the waste per bin deviation - there are large swings amongst the utilization of the bins.

6 Multi Dimensional Bin Packing, Applications, and Related Work

In this section we discuss some applications and related work. As many applications of bin packing tend to use some form of a general multidimensional format (in the literature this is sometimes referred to as GBPP) we first give a brief overview of some of those variants.

Bin packing's perhaps most literal application is packing real three dimensional physical objects like manufactured parts or separate items from an online purchase into a box for shipping. This is what is typically meant by the multidimensional bin packing problem (MDBPP). In its general form it consists of packing hyperpolygons (frequently rectangular) into

[8] Subject to the database and Perl hash ordering.

Table 2: `php5` results

`php5`	FF	BF	SS
Total Bins	8	8	8
Difference from Prod.	2	2	2
Avg. Item Size/Bin Mean	39.66	40.08	40.08
Avg. Item Size/Bin Stddev.	3.48	3.57	3.57
Total Waste	768	992	992
Waste/Bin Mean	96.00	124.00	124.00
Waste/Bin Stddev.	253.99	328.07	328.07
Largest Vhost 1.1x Increase Displacement	7	9	9
Largest Vhost 0.9x Decrease Displacement	39	30	30
Smallest Vhost 2x Increase Displacement	0	6	6
Smallest Vhost 5x Increase Displacement	0	6	6

Table 3: `php5` results

`php5`	FFD	BFD	SSD	BFD_LB
Total Bins	8	8	8	10
Difference from Prod.	2	2	2	4
Avg. Item Size/Bin Mean	67.43	67.43	67.43	101.55
Avg. Item Size/Bin Stddev.	64.36	64.36	64.36	89.83
Total Waste	992	992	992	4576
Waste/Bin Mean	124.00	124.00	124.00	457.60
Waste/Bin Stddev.	286.52	286.52	286.52	676.21
Largest Vhost 1.1x Increase Displacement	0	0	0	0
Largest Vhost 0.9x Decrease Displacement	2	2	2	2
Smallest Vhost 2x Increase Displacement	0	0	0	0
Smallest Vhost 5x Increase Displacement	0	0	0	0

hypercubes, again usually normalized to unit capacity in each dimension, though in some cases a single containing bin is treated to have unbounded height and the goal instead is to find the minimum height necessary to pack all the items into a rectangular hyperbox. [23]

One especially common version of this is that of packing 2D rectangles into a 2D bounding box. It comes in both orientable, where we are allowed to rotate the items, and non-orientable forms. This is often used in web site optimization of images so that rather than many individual resource round trip requests, a client need only suffer one such request. [33]

In that case, some common techniques include things like using FFD or BFD within strips, effectively choosing to optimize one dimension, say width, potentially at the expense of another, so that FFD is executed in iterative rounds starting from, say the lower left corner, and the strip height merely becomes the max height of an item placed during that round. Some optimizations can then be achieved by packing smaller items that will fit into the remaining space in the top right corner, again by FFD. [37]

The general multidimensional problem tends to use similar approaches: assigning a weighting to the various dimensions and running FFD, or another greedy heuristic, over each of them, or AI approaches like genetic algorithms, and iterated local search with multiple restart.

A slightly more restricted formulation of the multidimensional problem, and one that maps better to our particular example, is that of vector packing (MDVP). In it, each item is still represented as an array of sizes for each dimension, $\vec{i} =< s_{i_1}, ..., s_{i_d} >$, but rather than treating these as hyperboxes, we treat them as vectors, so that rather than filling out the corners of the bounding box in every dimension and at every level, each item placed reduces the problem to a new bounding box of size $\vec{B} - \vec{i}$. Viewed in the 2D case, we are placing rectangles so that the bottom-left corner of a new item touches the top-right corner of the previous one, or simply as placing vectors head to tail until we have reached any of the edges of any of our bounding box.

The vector packing problem maps very naturally to a number of resource allocation constraint problems since in a sense, every dimension represents an independently consumable resource, since each task (item) generally requires more than one resource, such as CPU, RAM, I/O, etc., and we cannot overrun our fixed server (bin) amount for any of them. As such it is also sometimes referred to as the multi-capacity bin packing problem (MCBPP).

One particularly interesting extension is task scheduling, since time can itself be viewed as a resource dimension. This idea has been applied to multiprocessor task scheduling by Johnson et al., though the problem differs slightly in that we have a fixed number of fixed sized bins rather than a variable number that we're trying to minimize. [6] [26] Instead the object is the place as many items into the bins (ie: schedule tasks on the cpus) as possible or to maximize the reward given by placing certain items.

Another common example these days, and one that is closely related to our example, is VM placement and consolidation strategies within a virtualization infrastructure or small cloud environment. Many different approaches have been considered to address this problem such as genetic algorithms [36] [38], LP rounding [**?**], iterated local search with multiple restart [31], and of course several variations on the greedy heuristics we have discussed thus far [32] [25].

However, as near as we can tell, bin packing is typically not directly used in production [18].

One reason is that although VMs are typically sized to fixed resource dimensions (eg: 2vCPUs and 4GB RAM) ahead of time by an administrator, actual workloads (ie: item sizes) vary quickly within such an environment and most solutions do not consider repacking. [9] Furthermore, there is a cost associated with migrating a VM and the result of doing so on the VMs already residing on the target location are not necessarily known ahead of time. Some approaches to bin packing [34] can take maximizing a profit (or minimizing a cost) value p_{ij} associated with packing each item into a particular bin, but in the context of VM placement those profit (or cost) values become functions whose values may change with every item placed making their applicability less obvious.

A second reason is that the heuristics involved in bin packing do not usually take load balance into account, though they may achieve some limited balance as a by product of the particular heuristic used. This is a concern we have attempted to address in our case study evaluations to a limited degree and would be worth further study.

Rather, implementations such as VMWare appear to be more concerned with approximating a load balancing solution by means of an online greedy hill climbing approach that iteratively finds the best VM to place or migrate within the fixed set of machines it

[9] See [19] for some theoretical results that do consider repacking and [31] for some application results with respect to the Machine Reassignment Problem (MRP).

has according to some cost-benefit metrics, and consider the use of bin packing as either an offline task to determine how large to size physical infrastructure during purchasing phases or to only be used during admission control phases when deciding whether or not to allow new VMs to be added to the running system - very similar to the mutliprocessor scheduling task, though even there there are differences since for instance not all physical servers may be identical.

However, as energy concerns continue to become more of a first-class citizen in such environments, as outlined for example in [5] [38], we believe that real time multi capacity bin packing approaches to determine the minimal number of physical servers necessary to service the current load requirements will play a larger role.

7 Conclusions and Future Work

In this paper we have presented a survey overview of the bin packing problem, some of its more important theoretical results, greedy heuristics and their analyses, and its applications to things like VM placement and consolidation.

We also conducted a brief evaluation of some of the heuristics in the context of a specific application case study: packing virtual hosts into VMs.

Overall we believe that while they are useful for sizing the raw number, they make poor assignment routines when things like load balance and displacement are taken into account. This is a feature that some industry applications seem to have acknowledged as well.

Some questions for future work we would like to consider include:

- Since our bin packing algorithm is only as good as our vhost size estimates, at what historical time scale or exponential factor should we process our logs at in order to give appropriate weight to historical versus recent data?
- Can we extend our sizing techniques to threaded Worker MPM type vhosts?
- Can we extend the bin packing to a 2 dimensional vector packing, provided we are able to find a way to measure actual CPU capacity and requirements accurately? Does this extra dimension perhaps account for the discrepancy between our predicted and production values?
- What is the measured impact that taking these sizing and placement adjustments into account have on our average response time latency or other workload performance metrics?

References

[1] S. Albers and M. Mitzenmacher. Average-case analyses of first fit and random fit bin packing. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 290–299. Society for Industrial and Applied Mathematics, 1998.

[2] D. L. Applegate, L. Buriol, B. Dillard, D. S. Johnson, and P. W. Shor. The cutting-stock approach to bin packing: theory and experiments. *Proceedings of Algorithm Engineering and Experimentation (ALENEX)*, pages 1–15, 2003.

[3] S. Bekman and E. Cholet. *Practical modperl*. O'Reilly Media, 2003.

[4] A. Borodin. *Online computation and competitive analysis*. Cambridge University Press, Cambridge, U.K. New York, 1998.

[5] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira. Effective vm sizing in virtualized data centers. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 594–601. IEEE, 2011.

[6] E. G. Coffman, Jr, M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[7] E. Coffman Jr, C. Courcoubetis, M. Garey, D. Johnson, P. Shor, R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, part i: Perfect packing theorems and the average case behavior of optimal packings. *SIAM Journal on Discrete Mathematics*, 13(3):384–402, 2000.

[8] E. Coffman Jr, C. Courcoubetis, M. Garey, D. S. Johnson, L. A. McGeoch, P. W. Shor, R. R. Weber, and M. Yannakakis. Fundamental discrepancies between average-case analyses under discrete and continuous distributions: A bin packing case study. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 230–240. ACM, 1991.

[9] E. Coffman Jr, D. Johnson, P. Shor, and R. Weber. Markov chains, computer proofs, and average-case analysis of best fit bin packing. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 412–421. ACM, 1993.

[10] C. Courcoubetis and R. Weber. A bin-packing system for objects with sizes from a finite set: Necessary and sufficient conditions for stability and some applications. In *Decision and Control, 1986 25th IEEE Conference on*, volume 25, pages 1686–1691. IEEE, 1986.

[11] C. Courcoubetis and R. Weber. Necessary and sufficient conditions for stability of a bin-packing system. *Journal of applied probability*, pages 989–999, 1986.

[12] C. Courcoubetis and R. Weber. Stability of on-line bin packing with random arrivals and long-run-average constraints. *Probability in the Engineering and Informational Sciences*, 4(4):447–460, 1990.

[13] J. Csirik, D. S. Johnson, and C. Kenyon. On the worst-case performance of the sum-of-squares algorithm for bin packing. *arXiv preprint cs/0509031*, 2005.

[14] J. Csirik, D. S. Johnson, C. Kenyon, J. B. Orlin, P. W. Shor, and R. R. Weber. On the sum-of-squares algorithm for bin packing. *Journal of the ACM (JACM)*, 53(1):1–65, 2006.

[15] J. Csirik, D. S. Johnson, C. Kenyon, P. W. Shor, and R. R. Weber. A self organizing bin packing heuristic. In *Algorithm Engineering and Experimentation*, pages 250–269. Springer, 1999.

[16] W. F. De La Vega and G. S. Lueker. Bin packing can be solved within 1+ ε in linear time. *Combinatorica*, 1(4):349–355, 1981.

[17] A. Federgruen and G. van Ryzin. Probabilistic analysis of a generalized bin packing problem and applications. *Operations research*, 45(4):596–609, 1997.

[18] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu. Vmware distributed resource management: Design, implementation, and lessons learned. *VMware Technical Journal*, 1(1):45–64, 2012.

[19] K. Jansen and K.-M. Klein. A robust afptas for online bin packing with polynomial migration. *arXiv preprint arXiv:1302.4213*, 2013.

[20] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974.

[21] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[22] N. Karmarkar and R. M. Karp. *The differencing method of set partitioning*. Computer Science Division (EECS), University of California Berkeley, 1982.

[23] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 289–298. ACM, 1984.

[24] C. Lee and D.-T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

[25] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder. Validating heuristics for virtual machines consolidation. *Microsoft Research, MSR-TR-2011-9*, 2011.

[26] W. Leinberger, G. Karypis, and V. Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, pages 404–412. IEEE, 1999.

[27] F. M. Liang. A lower bound for on-line bin packing. *Information processing letters*, 10(2):76–79, 1980.

[28] Y. Liu, N. Bobroff, L. Fong, S. Seelam, and J. Delgado. New metrics for scheduling jobs on cluster of virtual machines. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1001–1008. IEEE, 2011.

[29] E. C. man Jr, M. Garey, and D. Johnson. Approximation algorithms for bin packing: A survey. *Approximation Algorithms for NP-Hard Problems*, pages 46–93, 1996.

[30] S. Martello. *Knapsack problems : algorithms and computer implementations*. J. Wiley & Sons, Chichester New York, 1990.

[31] R. Masson, T. Vidal, J. Michallet, P. H. V. Penna, V. Petrucci, A. Subramanian, and H. Dubedout. An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. 2012.

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

[33] M. Perdeck. Fast Optimizing Rectangle Packing Algorithm for Building CSS Sprites. `http://www.codeproject.com/Articles/210979/Fast-optimizing-rectangle-packing-algorithm-for-bu`, 2011. [Online; accessed 19-April-2013].

[34] H. Shachnai and T. Tamir. Approximation schemes for generalized 2-dimensional vector packing with application to data placement. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 165–177. Springer, 2003.

[35] P. W. Shor. The average-case analysis of some on-line algorithms for bin packing. *Combinatorica*, 6(2):179–200, 1986.

[36] D. Wilcox, A. McNabb, K. Seppi, and K. Flanagan. Probabilistic virtual machine assignment. In *CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 54–60, 2010.

[37] S. Wong, Martin and Zhang. Survey on two-dimensional packing. `http://cgi.csc.liv.ac.uk/~epa/surveyhtml.html`, 2006. [Online; accessed 19-April-2013].

[38] J. Xu and J. A. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188. IEEE, 2010.

[39] A. C.-C. Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227, 1980.