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

Brian Kroth

2013-05-01

THE UNIVERSITY

Outline

Outline

The Problem

Basic Description

Like it sounds ... packing items into bins :)

Examples

- Physical Objects into boxes (manufacturing, shipping, etc.)
- Resource Allocation (especially VMs)
- Task Scheduling (with time constraints, though slightly different)

The Problem

Bin Packing - Formal Definition

- Want to pack a list of n items of varying sizes s_i into as few fixed sized bins of capacity B as possible.
- Sizes might be multi dimensional vectors ($< s_{i_1}, ..., s_{i_d} >$).
- Usually, normalize item sizes to $s_i \in (0, 1]$ and unit bin capacity of $B = 1$.

Bin Packing - Linear Program

Linear Program Formulation

- We can express it as a linear program:

$$\begin{array}{rlr} \text{Minimize:} & \sum_{j=1}^{n} y_j & \text{number of bins} \\ \text{Subject to:} & & \\ \text{bin capacity constraint} & \sum_{i=1}^{n} s_i x_{ij} \leq B y_j, & \forall\, j \in \{1, \ldots, n\}, \forall\, d \\ \text{one bin per item} & \sum_{j=1}^{n} x_{ij} = 1, & \forall\, i \in \{1, \ldots, n\} \\ & x_{ij}, y_j \in \{0, 1\} & \text{include item } i \text{ in bin } j \end{array}$$

- But, it's NP hard, even in the single dimensional case, so

Approaches

Classifications

- Greedy, (I)LP, AI (GA, ILS, Hill Climbing, etc.)
- Online and Offline
- Bounded and Unbounded Space

Analyses

Approaches

- Traditionally, worst case analysis, usually by constructing a worst case list where we can still compute OPT, and arguing via competitive analysis and careful accounting.
- Now, average case analysis, since many of them do fairly well most of the time.
  Examine things like expected waste on an algorithm on a given list distribution: $\mathbb{E}[W_{A(L)}]$.
  Problem: Continuous case doesn't model the real world problems very well, which have finite lists and often integer sizes.

Theoretical Results

Worst Case Bounds

- Yao showed no online algorithm can do better than $\frac{3}{2}OPT$.
- Split list into 3 parts $L_{\frac{1}{6}} L_{\frac{1}{3}} L_{\frac{1}{2}}$ with some ϵ splay.
- Can't be optimal on one part without giving ground on another.

Theoretical Results

Average Case Bounds

Courcoubetis and Weber used LP to show that discrete distributions can be classified by a convex cone to be in one of three cases:

1. $\exists$ a poly. online randomized algorithm OPT s.t. $\mathbb{E}[W_{OPT}] = O(1)$.
   So called perfect packing
2. $\exists$ OPT s.t. $\mathbb{E}[W_{OPT}] = \Theta(\sqrt{n})$.
3. $\exists$ OPT s.t. $\mathbb{E}[W_{OPT}] = \Theta(n)$.

Forms the basis for lots of other average case arguments since then you can potentially say what OPT will do in order to compare your algorithm against that.

Theoretical Results

Average Case Bounds

- The algorithm depends on finding solutions to LP.
- Generally NP-hard to determine which case your distribution is in.
- Coffman et al. showed that for $U\{j,k\} = \{\frac{1}{k}, \ldots, \frac{j}{k}\}$ we can apply the Perfect Packing Theorem and show that we're in case 1.
- They also showed how to derandomize the algorithm presented by Courcoubetis and Weber.

Outline

First Fit

Description

- Keep a fixed ordered list of currently "open" bins.
- When a new item arrives, place it in the first "feasible" bin in that list, else place it in a new bin.

Analysis

- Online
- Unbounded space
- $O(n \log n)$
- $\frac{17}{10}$ worst case competitive ratio.

First Fit Decreasing

Description

- First, sort the input list by decreasing sizes (1d).
- Apply FF

Analysis

- Offline
- Unbounded space
- $O(n \log n)$
- $\frac{11}{9}$ worst case approximation ratio.

Next Fit

Description

- Rather than an ordered list of open bins, just keep 1, or maybe k.
- Apply FF on that bin. When an arriving item won't fit, close that bin.

Analysis

- Online
- Bounded space (1 or k)
- $O(n)$
- 2 worst case competitive ratio. Consider a list of $2m$ repetitions of $< \frac{1}{2}, \frac{1}{2m} >$.

Refined First Fit

Description

- Proof for $\frac{17}{10}$ bound on FF relies on constructing a list within certain ranges and assigning a cost to packing each of those.
- RFF (Yao) classifies items by fixed ranges (4) and places an arriving item into a bin for that class using NF: $(\frac{1}{2}, 1]$, $(\frac{1}{3}, \frac{2}{5}]$, $(\frac{1}{3}, \frac{2}{5}]$, $(0, \frac{1}{3}]$
- Certain smaller items are also placed into bins for larger items to fill in the gaps.

Refined First Fit

Analysis

- Online
- Bounded space (4)
- $O(n)$
- $\frac{5}{3}$ worst case competitive ratio.

Harmonic

Description

- Extends RFF to M (a parameter) Harmonic ranges: $I_k = (\frac{1}{k+1}, \frac{1}{k}]$ for $k \in (1, M]$ and $(0, \frac{1}{M}]$ for I_M
- Exactly k items will fit into I_k when it is full.
- NF on each bin type, so only M in total necessary to keep open.
- Number of bins is invariant to arriving order of items!
- Can be extended to a "Refined" version like RFF so that certain smaller items are also placed into bins for larger items to fill in the gaps.

Harmonic

Analysis

- Online
- Bounded space (M, though 12 is sufficient)
- $O(n)$
- 1.6359 worst case competitive ratio (for Refined Harmonic).

Best Fit

Description

- Keep an ordered list of currently "open" bins.
- When a new item arrives, place it in the bin that will have the least gap remaining (best fit for that item).
- If list of bins is ordered by remaining capacity, that should be the first "feasible" bin in that list.

Best Fit

Analysis

- Online
- Unbounded space
- $O(n \log n)$
- $\frac{17}{10}$ worst case competitive ratio, same as FF. However, it does slightly better than FF for several average cases.
- If we don't care to track where individual items are placed, then we can represent a packing by the number of bins of a given remaining capacity, since the order of the bins doesn't matter.
- Then an arriving item is like walking that configuration state space from one to the next: Markov Chains!

Best Fit Decreasing

Description

- First, sort the input list by decreasing sizes (1d).
- Apply BF

Analysis

- Offline
- Unbounded space
- $O(n \log n)$
- $\frac{11}{9}$ worst case approximation ratio. Also better than FFD in some average cases.

Sum of Squares

Description

- BF does really well with symmetric distributions since we are likely to find a perfect packing arrangement.
- The intuition behind SS is to mimic this behavior when we don't know the distribution of items ahead of time.
- Tries to keep the number of bins with the same remaining capacity roughly the same for each remaining capacity.
- For a fixed sum of variables, their sum of squares is minimized when they are as close to equal as possible.

Sum of Squares

Analysis

- Online
- Unbounded space
- $O(nB)$ - not currently known how to improve a complete scan of all the bin sizes each time and item arrives.
- 3 worst case approximation ratio, but average case is close to 1 when the distribution is in the $O(1)$ class of CW and $O(\sqrt{n})$ in the second case.
- Can also tune SS to a particular distribution F and even have SS^* switch between these tuned versions online.

Outline

Setup

Problem Setup

- Lots of different types (PHP, HTML, Moodle, etc.) of vhosts each running as their own separate web server instance (for security and other reasons).
- Each served from a matching type of VM, with identical and fixed configuration.
- Mostly idle, so many vhosts of the same type packed into a single VM.

The Problem

Main Problem

We would like to know, how many VMs do we need to serve all the vhosts of a particular type?

Other Concerns

- Can BP "balance" the assignment of vhosts across VMs?
- What about changes in vhost size?

Item Sizes

Vhost Size

- Mostly concerned with memory requirements (also don't have a good way to measure CPU requirements)
- (number of processes/threads required to service vhost's requests at a given moment) * (size of each of those processes/threads)
- = Worker Count * Worker Size

Item Sizes

Worker Size

- Only dealt with Prefork MPMs, which have a full process (called workers) per request and linger on for a while in case they need to handle more.
- First, we attempt to find the actual memory requirement of one of these processes (eg: RSS - SHMEM).
- Problem: every vhost is different. So, do we take an average, or a max, or ...
- For safety reasons we chose to upper bound our guess by using a maximum observed process size.

Item Sizes

Worker Count

- Could have been measuring this for all vhosts over time, but we weren't (only for a few - technique verified against those).
- Instead, wrote a simulator to process access logs and based on behavior described in the MPM source code, determine how many concurrent processes were active to service requests at 1 second intervals.
- Summarized the data on a per day per vhost basis.
- Summarized that data by an EWMA with a half life of 7 days into a single value for each vhost that represents the average number of workers active.
- Various other distributions considered (eg: mean+stddev to get an average upper bound).

Evaluation

Evaluation

Armed with our item size lists (vhost sizes) and the bin size (the amount of RAM we allocate each VM of that type), we determined the following values for FF(D), BF(D), SS, BFD_LB on two vhost type lists:

- Total predicted bins compared with the number used currently (which may be wrong).
- Total waste and average/stddev waste per bin.
- Average/stddev average item size per bin.
- Number of vhosts displaced when we increase/decrease the largest vhost by $\frac{1}{10}$ and increase the smallest one by 2x and 5x.

BFD_LB Heuristic

BFD_LB Heuristic

The BFD_LB heuristic was a simple attempt at extending the BFD algorithm with some extra load balancing rules:

- For vhosts that are so large that they need to be split up, don't put those pieces together. All the other algorithms would happily do this.
- Don't place more than 1 item of size $\frac{1}{4}$ bin size into the same bin.

`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

php5	FF	BF/SS	FFD	BFD/SSD	BFD_LB
Total Bins	8	8	8	8	10
Difference from Prod.	2	2	2	2	4
Avg. Item Size/Bin Mean	39.66	40.08	67.43	67.43	101.55
Avg. Item Size/Bin Stddev.	3.48	3.57	64.36	64.36	89.83
Total Waste	768	992	992	992	4576
Waste/Bin Mean	96.00	124.00	124.00	124.00	457.60
Waste/Bin Stddev.	253.99	328.07	286.52	286.52	676.21
Largest Vhost 1.1x Increase Displacement	7	9	0	0	0
Largest Vhost 0.9x Decrease Displacement	39	30	2	2	2
Smallest Vhost 2x Increase Displacement	0	6	0	0	0
Smallest Vhost 5x Increase Displacement	0	6	0	0	0

Conclusions

Conclusions

- `moodle` results somewhat uninteresting since they pack everything into a single bin, so there's no difference between the standard heuristics.
- Our BFD_LB is a little closer to what we use in practice based on our added constraints.
- Seems to speak most to an inaccurate guess on the size of an individual worker process.

Conclusions

Conclusions

- `php5` results seem to show, based on avg item size stddev and waste/bin stddev that the BFD_LB method actually balances items worse than the semi random arrangement that the online heuristics give.
- This also naturally leads to more displacement when one of those items changes in size.

Outline

BP with Extra Dimensions

BP with Extra Dimensions

- Multi Dimensional BP (MDBP) concerned with packing hyperboxes into all of the corners of a hypercube.
  Like shipping or manufacturing context. Also image packing. Orientable or not. Sometimes with profit extensions.
- Vector Packing (MDVP) or Multi Capacity Bin Packing (MCBP)
  Each dimension is like a separately consumable resource. Means stacking items corner to corner, or head to tail like vectors.

Approaches to Extra Dimensions in BP

Approaches

- Greedy Heuristics still work, but now we need to weight the dimensions.
  Might be context dependent (eg: strip packing for images), or we could use something like the dot product in the context of MDVP.
- LP approaches still possible, though rounding becomes even harder.
- Genetic Algorithms, Hill Climbing, and other AI based approaches become more common. Mixed reviews on their effectiveness. Probably fairly application specific.

Task Scheduling

Task Scheduling

- Consider duration constraints as another packing dimension - MCBP?
- Not quite.
- Originally used in the context of multiprocessor scheduling, but there the number of "bins" is fixed.

VM Packing

VM Packing

- At the level below our case study.
- Concerned with how many VMs we can pack into a physical host.
- Number of hosts is usually fixed, so also not quite like bin packing.
- But can be used for admission control, same for task scheduling.
- Also useful for energy saving when we can reduce the number of powered on physical hosts needed to service VMs.

VM Packing

VM Packing

- Key issue: measuring and reacting to varying load.
- VMs are configured with static resource requirements ahead of time, but the demand on those resources is usually much less, so VMs are typically oversubscribed.
- So, like our case study, determining and reacting to load to maintain appropriate balance becomes a larger concern.
- In practice, hill climbing approach used to determine when and where to migrate VMs based on some cost/benefit analysis, not BP.
- BP just used for admission control and possibly purchasing sizing.

Questions?

Questions?