Server Consolidation and Bin Packing

Olivier Beaumont

Bin Packing and Server Consolidation

Joint Work With
Lionel Eyraud-Dubois, Hubert Larchevêque
Cepage – Inria Bordeaux Sud-Ouest – France
and
Christopher Thaves-Caro
LADyR, GSyC, Universidad Rey Juan Carlos, Madrid, Spain

0 INTRODUCTION

Introduction

General Setting

- Set of resources (PMs) to run services
- Set of services running as Virtual Machines (VMs)
- Placement of Virtual Machines (VM) onto a set of Physical Machines (PM) in a Cloud Computing platform.
- We study this problem on a provider-side $\longrightarrow$ optimize the use of resources (minimize the number of used PMs).

Consolidation

- Each Virtual Machine comes with resource demands : CPU, memory, bandwidth, I/O,...
- How to place VMs onto PMs so as to minimize the number of resources.
- In fact the user has paid for resources, and SLA has to be enforced at any moment if the VMs really needs it.
- In practice, the real use of the reserved resources is quite low.
- Using those reserved resources that are not use to run other Virtual Machines, the global resource utilization can be improved.

Bin Packing

- Several dimensions : memory, IO, network, CPU,...
- Multi dimensional vector packing $\Longrightarrow$ very hard to approximate
- $+$ online setting (new reservations, new machines, failures)
- $+$ size of the items can change over time (under utilization of the reservation)
- $+$ placement constraints (close or far away) between VMs
- $+$ many more

Is there a chance ?

- Can we rely on something else than cheap, reasonable and clever heuristics on one extreme
- ... or a constraint satisfying solver on the other extreme !
- Is there a way to model the system
  - so as to retain the main features
  - while keeping the problems tractable ?
- Three different options :
- characteristics of applications that could be used to ease the problem.

Example 1 : Splittable Items

- A certain number of services to run (all the time)
- or a certain number of requests (web server) to handle simultaneously :
- these services can be served by several machines.
- Nevertheless, not all demands split identically
  - But if CPU, bdw, IO can indeed be split, memory cannot.
  - N services executed K PMs each running n_k instances
  - the memory footprint $\text{mem}_k = \text{mem}$
  - but $\sum \text{cpu}(i) = \text{cpu}$ is enough (same for I/O, bandwidth,...)
- this is not a multi dimensional packing problem

Example 2 : Dynamicity

- VMs come with a demand that may change over time.
- Idea $\leftrightarrow$ trade approximation ratio against robustness
- Restrictions
  - non splittable services
  - mono-dimensional criterion (say CPU)
- If you lower the requirements in terms of approximation ratio, then it is easier to maintain a (bad) solution at low cost

Example 3 : Robustness Issues

- Several Services running as independent VMs.
- Service S_i needs (at least) k_i VMs
- m machines, capacity c, failure probability (next day) p
- solution: $n_{i,j}$ services for A_i on Machine M_j
- such that probability (after one day) that $\sum n_{i,j} \geq k_i$ is at least p_i.

1 SPLITTABLE ITEMS

Example 1 : Splittable Items [IEEE TPDS 2012]

- Several services to run.
- Each service can be served by any number of machines.
- Each service has an overall CPU demand w_j
- and an homogeneous normalized memory footprint of 1.
- Each PM has an overall CPU capacity b_i
- and an overall memory d_i.

Rationale

- The overall demand w_j can be split over several PMs.
- Each time we start a service on a PM, remove 1 from its memory.
- The CPU capacity of the PM b_i can be shared between services.

Precise model

An instance

- m PMs, with CPU capacity b_i and memory d_i
- n services, with demand w_j

A solution

- An assignment w_j^i of service i to PM j
- $\forall j, \quad \sum_i w_i^j \leq b_j$ (capacity constraint at PM j)
- $\forall j, \quad \text{Card}\{i, \quad w_j^i > 0\} \leq d_j$ (memory constraint at server j)
- $\forall i, \quad \sum_j w_i^j \leq w_i$ (demand constraint for service i)

Complexity

- NP-Hard: reduction from 3-Partition
  - $3n$ items of size $\frac{B}{4} < a_i < \frac{B}{2}$, $\sum a_i = nB$
  - is there a partition into n groups of 3 elements suming up to B
  - n PMs with capacities B and memory 3
  - $3n$ services with demand a_i, $\sum a_i = nB$
  - Throughput nB reachable iff 3-Partition has a solution
- Easy to solve without the memory constraint
- Seems much easier if you slightly relax the memory constraint

$\rightarrow$ Loosen the memory constraint (will be solved later).

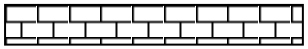
Algorithm Seq

- Resource augmentation: allow one more memory unit per PM
- Order services by increasing demand
- For each PM, with CPU capacity b and memory d:
  1. Find a consecutive sublist of length $d+1$ such that:
     - total demand is at least b
     - demand of the first d services is less than b
  2. Assign these services, perhaps split the last one
  3. Update the service list
- Choice of a subset does not affect optimality
- Ordering of the PMs does not affect optimality

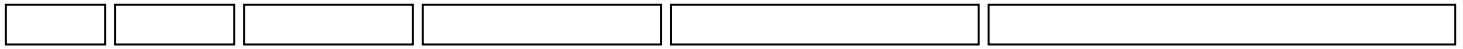
An example

$d_3 = 1$
$b_3 = 68$

$d_2 = 5$
$b_2 = 30$



$d_1 = 2$
$b_1 = 48$

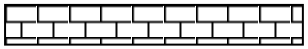


10 12 17 $w_4 = 24$ 31 47

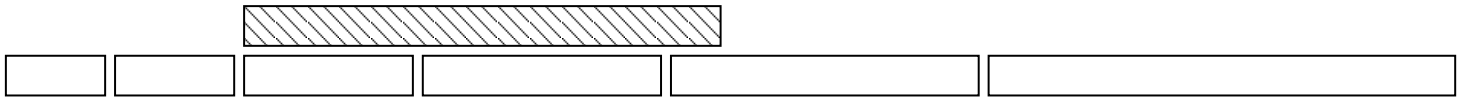
An example

$d_3 = 1$
$b_3 = 68$

$d_2 = 5$
$b_2 = 30$



$d_1 = 2$
$b_1 = 48$

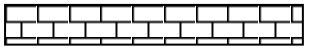


10 12 17 $w_4 = 24$ 31 47

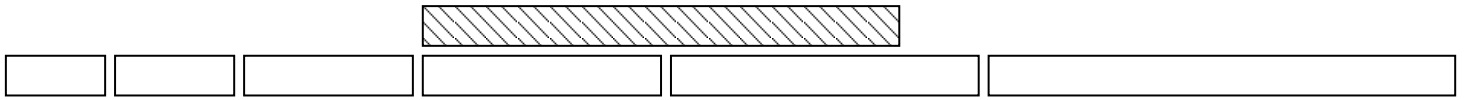
An example

$d_3 = 1$
$b_3 = 68$

$d_2 = 5$
$b_2 = 30$



$d_1 = 2$
$b_1 = 48$

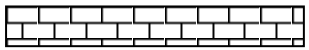


10 12 17 $w_4 = 24$ 31 47

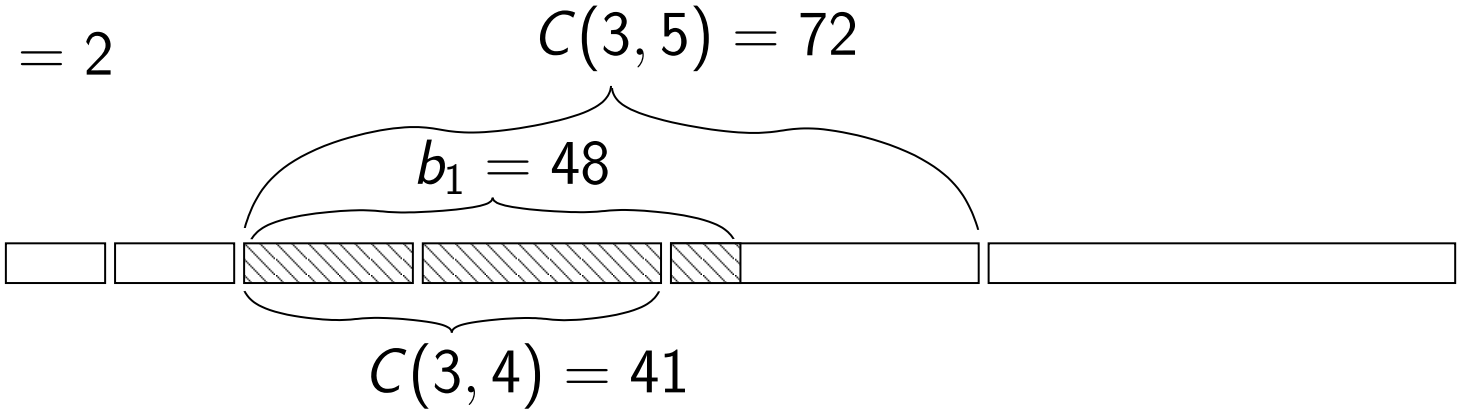
An example

$d_3 = 1$
$b_3 = 68$

$d_2 = 5$
$b_2 = 30$

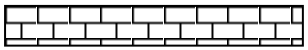


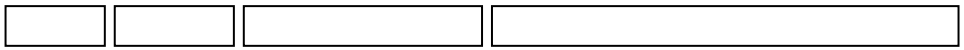
$d_1 = 2$



$d_3 = 1$
$b_3 = 68$

$d_2 = 5$
$b_2 = 30$





10 12 24 47

$d_1 = 2$
$b_1 = 48$

An example

$d_3 = 1$
$b_3 = 68$

$d_2 = 5$
$b_2 = 30$

10 12 24 47

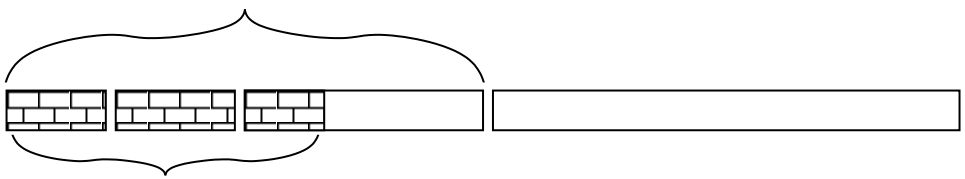
$d_1 = 2$
$b_1 = 48$

An example

$d_3 = 1$
$b_3 = 68$

$d_2 = 5$

$C(1,3) = 46$

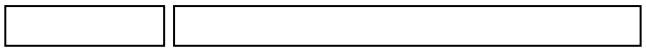


$b_2 = 30$

$d_1 = 2$
$b_1 = 48$

An example

$d_3 = 1$
$b_3 = 68$



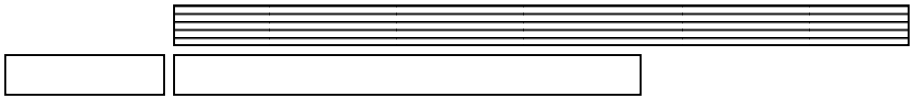
16 47

$d_2 = 5$
$b_2 = 30$

$d_1 = 2$
$b_1 = 48$

An example

$d_3 = 1$
$b_3 = 68$



16 47

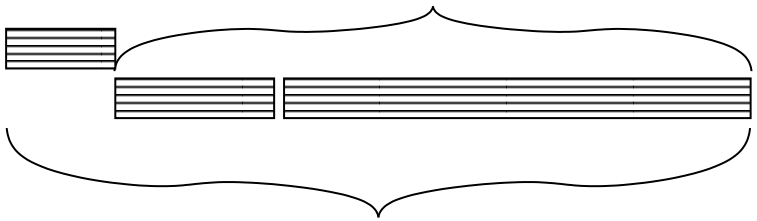
$d_2 = 5$
$b_2 = 30$

$d_1 = 2$
$b_1 = 48$

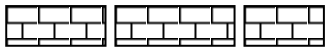
An example

$d_3 = 1$

$C(1,2) = 63$



$b_3 = 68$



$d_2 = 5$

$b_2 = 30$

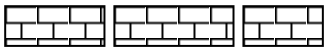
$d_1 = 2$

$b_1 = 48$

An example

5 Remaining server

$d_3 = 1$
$b_3 = 63$



$d_2 = 5$
$b_2 = 30$

$d_1 = 2$
$b_1 = 48$

Why does it work ?

Intuitively, more disparate service demands are "easier" to allocate

Central Lemma

Define $\mathcal{C} \preceq \mathcal{D}$ iff $\forall k, \sum_{i=1}^{k} C_i \leq \sum_{i=1}^{k} D_i$

Statement: if $\begin{array}{ccc} \mathcal{C} & \xrightarrow{\text{SEQ}(d+1,b)} & \mathcal{C}' \\ \preceq & & \\ \mathcal{D} & \xrightarrow{\text{valid}(d,b)} & \mathcal{D}' \end{array}$ then $\mathcal{C}' \preceq \mathcal{D}'$

- Recursively, $\mathcal{C}^{(m)} \preceq \mathcal{D}^{(m)}$, thus $\sum C_i^{(m)} \leq \sum D_i^{(m)}$
- Remaining service capacity is lower with SEQ than with any *valid* allocation

Remarks

Valid approximation algorithm

- In order to obtain a valid scheme,
- start with mem $\longleftarrow$ mem -1.

Dual problems

- Given a set of services, minimize the memory d^* needed to execute them
- SEQ with dichotomy achieves $d^* + 1$

Simple heuristics

Largest Service Largest PM

Order services and PMs by capacities, and assign the currently largest service to the currently largest machine. Split and reinsert the service if partially served.

Largest Service Best Connection

Same as before, but sort PMs by $\frac{b_j}{d_j}$ (average available capacity per memory unit).

Online Best Connection

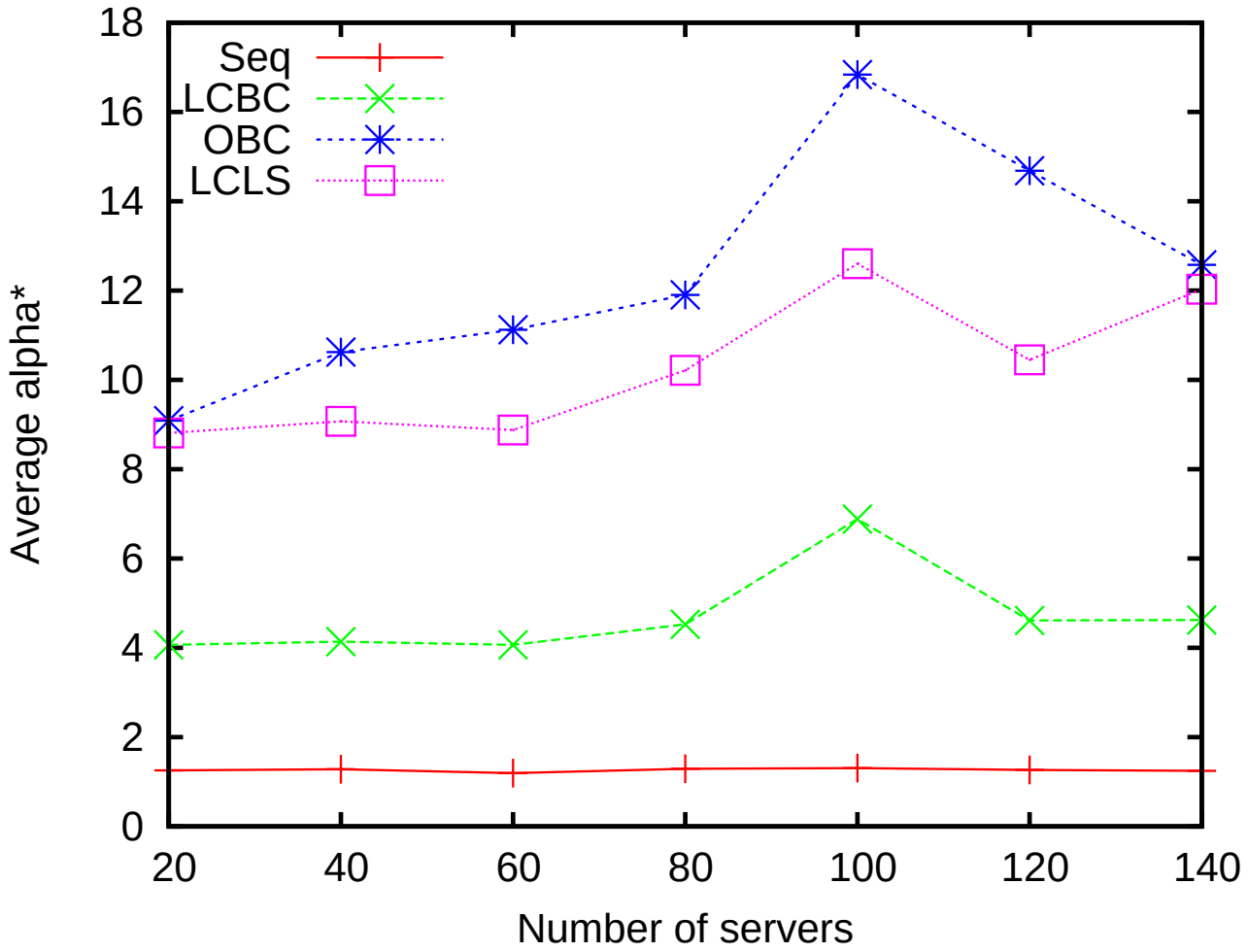
Same as LCBC, but without sorting services first. Use the PM with the closest available capacity per memory unit from the considered service

Experimental setting

Random instance generation

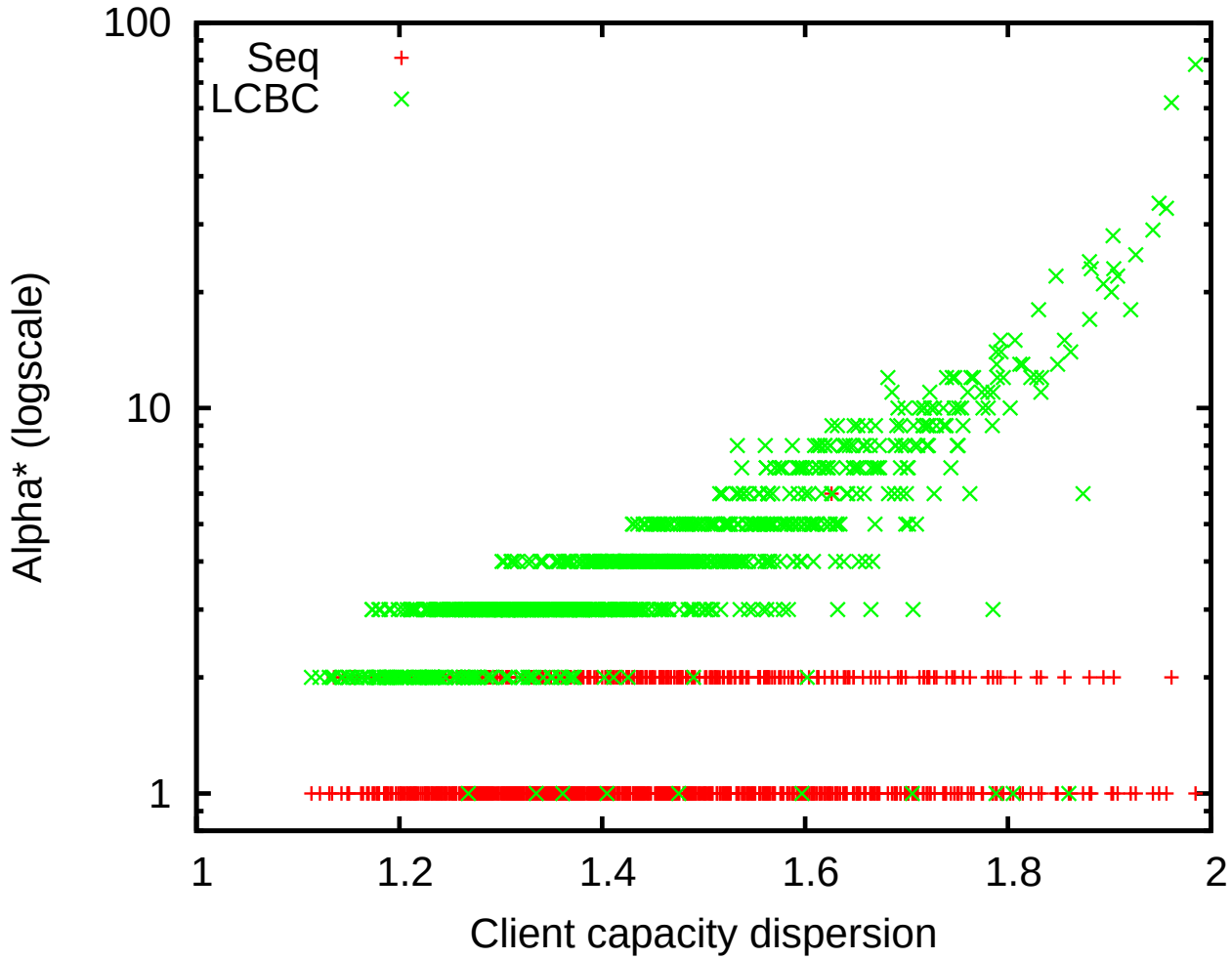
- m PMs, $10m$ services
- Capacities generated with power law distributions
- PM memories nearly proportional to their CPU capacities

Results



Average α^* when m varies

Results



α^* values against dispersion for $m = 80$

Can we do it online ?

When new PMs can be added

- PMs can be considered in any order
- and so, the algorithm is fully online with respect to PM addition.

When services come and go

- considering sorted lists of services is crucial...
- Count the number of changes for various algorithms.

Fully online is impossible

There is no fully online algorithm with resource augmentation factor α and approximation ratio $\frac{1}{k}$.

- 1 PM with capacity $b = k \times 2^{\alpha k+1}$ and memory k
- αk groups of services, group i having capacity 2^i
- one client of demand b

- $\mathcal{A}$ must connect at least one client from each group.
- $\Rightarrow$ No more connection available for the last client.

Online SEQ

Add some "locality" in SEQ

- Always choose the "rightmost" sublist of services
- $\Rightarrow$ Ensures that the split service is reinserted at the same place in the list

Local transformations of service lists

- $\mathcal{C}$ is *increased* to $\mathcal{C}^+$ by
  - insertion of a new service at position p
  - capacity increase of $\mathcal{C}_{p+1}$
- Similarly, $\mathcal{C}$ is *decreased* to $\mathcal{C}^-$ by
  - deletion of a service at position p
  - capacity decrease of $\mathcal{C}_{p+1}$

Online SEQ

Lemma

If $\mathcal{C} \xrightarrow{\text{SEQ}(d+1,b)} \mathcal{C}'$, $\mathcal{C}^+ = \mathcal{D}$, then $\mathcal{C}' \xrightarrow{+} \mathcal{D}'$ and $\mathcal{D} \xrightarrow{\text{SEQ}(d+1,b)} \mathcal{D}'$

Furthermore, the allocations differ by at most 4 changes.

Recursively, for a given set of PMs $\mathcal{S}$, $\text{SEQ}(\mathcal{C} \cup \mathcal{C}_{\text{new}})$ and $\text{SEQ}(\mathcal{C})$ differ by at most 4 changes per server.

A comparison

Aggressive Best Connection

- On service arrival, connect with Best Connection. If no room, remove the client that yields the largest gain.
- On client departure, use the newly available bandwidth to reduce the indegree of other clients. If there are not connected clients left, act like on client arrival.

On 80-server instances, with 500 events

- Maximal number of changes with the online heuristic for one event can reach 130 for one server (4 in our case)
- Average number of maximal changes is 3.5 for SEQ, 1.6 for ABC
- Which is the best ?

2 Dynamicity

Dynamicity

- VMs come with a demand that may change over time.
- Restrictions
  - non splittable services
  - mono-dimensional criterion (say CPU)
- Idea $\leftrightarrow$ trade approximation ratio against robustness
- If you lower the requirements in terms of approximation ratio, then it is easier to maintain a (bad) solution at low cost

Known Results

- fully online settings – but migrations allowed $\longrightarrow$ bi-criteria problem (approximation ratio for BP, number of migrations)
- without migrations, not better than 2.43 [ICALP'05] (first fit is worse than 2.5)
- 1.25 approximation with a constant number of migrations [ESA'93]
- 1.33 approximation with a most 7 migrations per modification [SIAM'00] (cheating by moving many small VMs simultaneously)
- Here : 1.5 approximation (valid solution after one migration only + local reconfigurations afterwards)

Basic Idea : Classification

- Classification of the objects
- $0 < T \leq \frac{1}{3} < S \leq \frac{1}{2} < L \leq \frac{2}{3} < B \leq 1$
- Possible arrangements into a bin $B, LT*, S^2, T*$
- Also accept 1 S bin and all $T*$ bins have weight at least $\frac{2}{3}$ except 1 UT.
- Approximation ratios are asymptotic (or add 2).
- Algorithm : react when a bin becomes too big or too small.

Algorithm

- Says how to react to overfull bins
  - $BT* : T* \longrightarrow BT*.new(B)$;
  - $BBT* : T* \longrightarrow BT* \longrightarrow BBT*.new(B)$;
  - $BLT* : T* \longrightarrow LT* \longrightarrow BLT*.new(B)$;
  - $BLST* : T* \longrightarrow ST* \longrightarrow LST* \longrightarrow BLST*.new(B); insert(S)$;
  - $BS^2T* : T* \longrightarrow ST* \longrightarrow S^2T* \longrightarrow BS^2T*.new(B)$;
  - $BST* : T* \longrightarrow ST* \longrightarrow BST*.new(B)$;
  - $LLST* : T* \longrightarrow ST* \longrightarrow LST* \longrightarrow LLST*.new(L); fill(L); insert(S)$;
  - $LLT* : T* \longrightarrow LT* \longrightarrow LLT*.new(L); fill(L)$;
  - $LS^2T* : T* \longrightarrow ST* \longrightarrow S^2T* \longrightarrow LS^2T*.new(S^2)$;
  - ...
- and the same for underloaded bins...

Proof of correctness

- $0 < T \leq \frac{1}{3} < S \leq \frac{1}{2} < L \leq \frac{2}{3} < B \leq 1$
- If there is a $T*$ bin, then
- possible arrangements into a bin are $B, LT*, S^2, T*, 1S, 1UT*$
- all but 2 bins have weight $\frac{2}{3}$
- If not, consider the case without Ts : improve Opt, does not change $\mathcal{A}$.
  - $n(B) = n_{\text{opt}}(B)$
  - $n(L) = n_{\text{opt}}(L) + n_{\text{opt}}(LS)$
  - $2n(S^2) = n_{\text{opt}}(LS) + 2n_{\text{opt}}(S^2)$
  - summing up provides the $\frac{3}{2}$ approximation ratio

3 Robustness

Robustness Issues

- Assume that several services S_i are running
- Service S_i requires at least k_i identical VMs to run properly
- If one PM M_j fails with $n_{i,j}$ services for A_i
- then $n_{i,j}$ services should be restarted elsewhere (without state).
- Questions : if each machine comes with a failure probability p for the next day,
- how to allocate the VMs to PMs so as to guarantee that S_i will run the next day with probability p_i

Rationale

- Closed problems have been widely studied in the literature
- Rocket problem : embark enough replicas so as to be sure that enough will remain valid at the end of the journey
- Here, the same replica may even be used for several applications (just memory and CPU).
- Advantage : the problem is much easier (in terms of solution quality)
- that if you replace services by tasks !

4 Conclusions

Context

- Server Consolidation Problems can be seen as a Multi-Dimensional Vector Packing Problem
- with heterogeneous bins
- where items arrive online
- and robustness (SLA) is ensured contractually
- and placement of machines should preserve locality
- and...
- but it should NOT !

Summary

- Server Consolidation Problems have plenty of good features
- splittable items
- services running have no state
- lower utilization and migrations make the problems easier
- $\longrightarrow$ still plenty of room for approximation dedicated algorithms.

Is it so important after all ?

- If you are interested in processing time
- clever heuristics work well.
- If you are interested in quality
- constraint solvers are not that slow.
- What is important is rather to evaluate what helps (split items, trade between objectives, concentrate of classes of applications) and what is "too expensive" (replicate everything, rigid applications)
- what may be used as incentives by providers.