Placing Virtual Machines in a Cloud under Constraints

Fabien Hermenier — placing rectangles since 2006

@fhermeni
fabien.hermenier@nutanix.com
https://fhermeni.github.io
Gestion dynamique des tâches dans les grappes, une approche à base de machines virtuelles

How to design a better testbed: Lessons from a decade of network experiments

2006 - 2010
PhD - Postdoc

2011
Postdoc

2011 - 2016
Associate professor

VM scheduling, green computing
VM scheduling, resource management
Virtualization

Entreprise cloud company
“Going beyond hyperconverged infrastructures”
I am from the (distributed) system community not from the CP community
Inside a private cloud
Clusters

from 2 to $x$ physical servers

isolated applications

virtual machines
containers

storage layer

SAN based: converged infrastructure
shared over the nodes: hyper-converged infrastructure
monitoring data

VM queue

VM scheduler

cloud model

decisions

actuators
VM scheduling

find a server to every VM to run

Such that

compatible hw
enough pCPU
enough RAM
enough storage
enough whatever

While

min or max sth
A good VM scheduler provides

Bigger business value, same infrastructure
A good VM scheduler provides:

- Same business value,
- Smaller infrastructure.
A good **dynamic** VM scheduler fixes issues online.

- hotspot mitigation
- re-balancing
- dynamic packing
KEEP CALM AND CONSOLIDATE AS HELL

1 node =

VDI workload:
12+ vCPU/1 pCPU

100+ VMs / server
2005: Live-migration of VMs

temporary, resources are used on the source and the destination nodes
Migrations are costly
dynamic schedulers

dependency management

VM1
VM2
VM3
VM4
VM5
VM6

N1
N2
N3
N4

cpu
mem
dynamic schedulers

anti-affinity(VM3, VM4)
min(#onlineNodes)

anti-affinity(VM3, VM4)
min(#onlineNodes)
dynamic schedulers

cyclic dependencies

a pivot to break the cycle

fix or prevent the situation?
dynamic schedulers
quality at a price

sol #1: 1m, 2m, 2m

\[
\min(\#\text{onlineNodes}) = 3
\]
dynamic schedulers

quality at a price

min(#onlineNodes) = 3

sol #1: 1m, 2m, 2m

sol #2: 1m, 2m, 1m

lower MTTR (faster)
static schedulers

consider the VM queue deployed everywhere

fragmentation issues

dynamic schedulers

live-migrations to address fragmentation

Costly (storage, migration latency)

thousands of articles

over-hyped ?

but used in private clouds (steady workloads ?)
## Placement constraints

### customer or provider side

<table>
<thead>
<tr>
<th>various concerns</th>
<th>performance, security, power efficiency, legal agreements, high-availability, fault-tolerance …</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>spatial (placement) or temporal (scheduling)</td>
</tr>
<tr>
<td>enforcement level</td>
<td>hard or soft discrete or continuous</td>
</tr>
<tr>
<td>manipulated concepts</td>
<td>state, placement, resource allocation, action schedule, counters, etc.</td>
</tr>
</tbody>
</table>
discrete constraints

\[
\text{spread}(VM[1,2]) \\
\text{ban}(VM1, N1) \\
\text{ban}(VM2, N2)
\]

continuous constraints

spread(VM[1,2])
ban(VM1, N1)
ban(VM2, N2)

“simple” spatial problem

harder scheduling problem
(think about actions interleaving)
hard constraints

spread(VM[1..50])
must be satisfied
all or nothing approach
not always meaningful

soft constraints

mostlySpread(VM[1..50], 4, 6)
satisfiable or not
internal or external penalty model

harder to implement/scale

hard to standardise?
The constraint catalog evolves

- Dynamic Power Management (DRS 3.1) 2009
- VM-VM affinity (DRS) 2010
- Dedicated instances (EC2) mar. 2011
- MaxVMsPerServer (DRS 5.1) sep. 2012
- VM-host affinity (DRS 4.1) apr. 2011
- The constraint needed in 2014
- The constraint needed in 2016
min(x) or max(x)
atomic objectives

\[ \min(\text{penalties}) \]

\[ \min(\text{Total Cost Ownership}) \]

\[ \min(\text{unbalance}) \]

\ldots
composite objectives using weights

$$\min(\alpha x + \beta y)$$

How to estimate coefficients?
useful to model sth. you don’t understand?

$$\min(\alpha \text{TCO} + \beta \text{VIOLATIONS})$$

€ as a common quantifier:
$$\max(\text{REVENUES})$$
Optimize or satisfy?

min(...) or max(...)

easy to say

hardly provable

composable through weighting magic

threshold based

domain specific expertise

verifiable

composable
Acropolis Dynamic Scheduler mitigates hotspot

Trigger
- 15 min

Thresholds
- 85%
  - CPU
  - storage-CPU

Maintain
- affinity constraints
- Resource demand
  (from machine learning)

Minimize
- $\sum$ mig. cost

30 sec. timeout
128MB RAM max
adapt the VM placement depending on pluggable expectations

network and memory-aware migration scheduler, VM-(VM|PM) affinities, resource matchmaking, node state manipulation, counter based restrictions, energy efficiency, discrete or continuous restrictions
interaction though a DSL, an API or JSON messages

spread(VM[2..3]);
preserve(VM1,'cpu', 3);
offline(@N4);

The reconfiguration plan

0'00 to 0'02: relocate(VM2,N2)
0'00 to 0'04: relocate(VM6,N2)
0'02 to 0'05: relocate(VM4,N1)
0'04 to 0'08: shutdown(N4)
0'05 to 0'06: allocate(VM1,'cpu',3)
P.S: From a theoretical to a practical schedule

duration may be longer
convert to an event based schedule

/!/\ possible loss of quality but no alternative yet

0:3 - migrate VM4
0:3 - migrate VM5
0:4 - migrate VM2
3:8 - migrate VM7
4:8 - shutdown(N2)
- : migrate VM4
- : migrate VM5
- : migrate VM2
!migrate(VM2) & !migrate(VM4): shutdown(N2)
!migrate(VM5): migrate VM7
The right model for the right problem

**CHOCO**

An Open-Source Java library for constraint programming

deterministic composition
high-level constraints

\[ X = \{ x_1, x_2, x_3 \} \]

\[ D(x_i) = [0, 2], \forall x_i \in X \]

\[ C = \{ \]
\[ c_1 : x_1 < x_2 \]
\[ c_2 : x_1 + x_2 \geq 2 \]
\[ c_3 : x_1 < x_3 \]
From how it worked to how it works
Consequences of trying to understand
No knowledge in CP: Collaboration with the Nantes team

2 phases approach: 1 problem for the placement (VMPP),
1 for the “schedule” (VMRP)

(Faster & better than 1 phase + weighting magic)

[vee 2009]
The placement problem (VMPP)

VMs are idle or burning the CPUs

Old-school bin-packing

$H_i$: bit vectors + bool_channelling

$R_p \cdot H_i \leq C_p(n_i)$ $\forall n_i \in N$

$R_m \cdot H_i \leq C_m(n_i)$ $\forall n_i \in N$

Multi-knapsack constraint + dynamic programming
(Thanks Xavier Lorca & Hadrien Cambazard)

Symmetry breaking for items not fitting

Minimize #nodes using atmost_nvalue
(green computing hype)
The replacement problem (VMRP)

VMPP variation:

atmost_nvalue, \#nodes as a constant

migration_duration: [0, K], 0 iff stays

dependency_management & cost function: hard coded inside a propagator

min (sum of the end moment of migrations)
Symmetry breaking
50% faster

Global perf
2 minutes for a local opt.
Reviewing Entropy design

Scheduling heuristic prevents control
hard-coded but vital heuristic inside a propagator
not optimisation friendly

Scalability issue
model-level: n*m binary variables, 5k VMs/100 servers: 500k bool vars
memory level: see above (4GB RAM in 2006)

Objective issue
No one really cares about $\min(\#\text{nodes})$
$\min(\#\text{migration})$ dominates
coarse grain staging
delay actions

mig(VM2)
mig(VM4)
mig(VM5)
off(N2)
mig(VM7)

stage 1

stage 2
time
2010+ : let schedule properly
Resource-Constrained Project Scheduling Problem

Tasks to model resource consumption

Every action modelled from its impact over resource

Booting a VM:
Place a task on the node

Migrating a VM:
2 tasks (source and destination)
Tasks co-locate if the VM stays
(impact the overlapping period)
Original implementation

Using Choco “cumulativeS” constraint (1 per resource)
(Thanks Arnaud Malapert)
Regular cumulative constraints + binary variables to state
if the tasks is in.

Conditional non-overlapping modeled
outside the constraints

N1
<table>
<thead>
<tr>
<th>VM1</th>
<th>VM1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N2
<table>
<thead>
<tr>
<th>VM1</th>
</tr>
</thead>
</table>

N1
<table>
<thead>
<tr>
<th>VM1</th>
</tr>
</thead>
</table>

No more cyclic dependencies
Prevent the situation in place of fixing it. Should not happen anyway

Enough perf for a Poc. Not enough for being good enough
First implementation falls with more than 10 nodes
From cumulativeS to myCumulative

Problem: overkill filtering
2 kind of tasks: starts at 0 (c-task) or ends at makespan (d-task)

Problem particularities:
A VM migration is modelled using 1 c- and 1 d-task
no overlap when there is no migration

Moving to a home-made implementation
(thanks Sophie Demassey)
active only once the Items are placed
From myCumulative to myCumulative 2

Booting || halting the node:
No longer modelled using tasks

First implementation by my own
hackie, incorrect, big fail

Second implementation by Sophie Demassey
big win

Quick improvement by my side
semi-win: bug found 2 years later (over-filtering)
2011+ : let pack properly

From 1 bin packing per dimension
to one xD vector packing

Originally:
same filtering but less events, less memory
get rid of the knapsack filtering from profiling observation
no more bool vector. Useless.

Now:
knapsack is back again (see later).
start to consider cardinality dimension as a pivot

Un-effective tryout: static or dynamic big-items
BtrPlace becomes a CP solver dedicated to VM management

A primary problem to model a reconfiguration
Extensions to bring additional concerns
Constraints to manipulate
BtrPlace core CSP models a reconfiguration plan
1 model of transition per element
action durations as constants

\[ boot(v \in V) \triangleq \quad D(v) \in \mathbb{N} \]
\[ st(v) = [0, H - D(v)] \]
\[ ed(v) = st(v) + D(v) \]
\[ d(v) = ed(v) - st(v) \]
\[ d(v) = D(v) \]
\[ ed(v) < H \]
\[ d(v) < H \]
\[ h(v) \in \{0, \ldots, |N| - 1\} \]

\[ relocatable(v \in V) \triangleq \ldots \]
\[ shutdown(v \in V) \triangleq \ldots \]
\[ suspend(v \in V) \triangleq \ldots \]
\[ resume(v \in V) \triangleq \ldots \]
\[ kill(v \in V) \triangleq \ldots \]
\[ bootable(n \in N) \triangleq \ldots \]
\[ haltable(n \in N) \triangleq \ldots \]
Views bring additional concerns
new variables and relations

ShareableResource(r) ::= 
\[ \forall n \in \mathcal{N}, \sum_{v \in \mathcal{V}, host(v) = n} cons(v, r) \leq capa(n, r) \]

Network() ::= ...

Power() ::= ...

High-Availability() ::= ...
Constraints state new relations

\[
\text{spread}(X \subseteq \mathcal{V}) \triangleq \forall (a, b) \in X, \text{host}(a) \neq \text{host}(b)
\]

\[
\text{lonely}(X \subseteq \mathcal{V}) \triangleq \bigcup_{v \in X} \text{host}(v) \not\subseteq \bigcup_{v \in \mathcal{V} \setminus X} \text{host}(v)
\]

\ldots

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>spread</td>
<td>50</td>
<td>root</td>
<td>11</td>
<td>preserve</td>
<td>10</td>
</tr>
<tr>
<td>among</td>
<td>40</td>
<td>lonely</td>
<td>17</td>
<td>overSubscription</td>
<td>40</td>
</tr>
<tr>
<td>ban</td>
<td>20</td>
<td>quarantine</td>
<td>40</td>
<td>offline</td>
<td>10</td>
</tr>
<tr>
<td>fence</td>
<td>58</td>
<td>capacity</td>
<td>64</td>
<td>noIdles</td>
<td>10</td>
</tr>
<tr>
<td>gather</td>
<td>11</td>
<td>splitAmong</td>
<td>31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CP expressivity leads to concise code
2009 - 2011
FIT4green aims at contributing to ICT energy reducing efforts by creating an energy-aware layer of plug-ins for data centre automation frameworks.

2010: switch from their heuristic to Entropy/BtrPlace.
homemade (very fine grain) objective
Involved as a contributor
2.8k VMs, 700 servers
400 sec. to the 1st solution

“Houston (Fabien), we got a problem, CP is sloowww
95%+ in search heuristic
no value gettable $O(1)$, $O(n)$ at least

One cache later:
from 400 to 20 sec.
[adhoc network 12]

Lesson learned

1/ CP was not slow, they did not try to understand the issue
2/ The fix is not good/disruptive/innovative. Normal due to the context
3/ Their opinion did not change as they forgot about the fix.
2011: better knowledge of the problem

There is no anarchy in a data center load spike, failures: local issue

The data center is not a single space
Technical limitations create *autonomous regions*

Static instance analysis to the rescue
static model analysis 101

scheduler.doRepair(true)

manage only supposed mis-placed VMs
Pre-place “well placed VM”
beware of under estimations!
spread({VM3,VM2,VM8});
lonely({VM7});
preserve({VM1},'ucpu', 3);
offline(@N6);
ban($ALL_VMS,@N8);
fence(VM[1..7],@N[1..4]);
fence(VM[8..12],@N[5..8]);

s.setInstanceSolver(
    new StaticPartitioning())

independent sub-problems solved in parallel
beware of resource fragmentation!
Repair benefits

5 times less VMs to consider
10 times faster at least
no impact on the solution quality

Partitioning benefits

2 times smaller, 4 times faster
no impact on the solution quality
2011: The hype of set variables
(Better knowledge of the model & the solver)

In choco bin packing
- In the user-API to state the items in every bin
- Embed a channeling to the placement variables

In side constraints
- Lonely: 2 sets of VMs must be on disjoint set of nodes.
  using the set_disjoint constraint
[CP 2011, industry track]

Results

From 2 to 6k VMs on 1k servers

Table 1: Impact of the consolidation ratio on the solving process.

<table>
<thead>
<tr>
<th>Ratio</th>
<th>solved</th>
<th>obj</th>
<th>nodes</th>
<th>fails</th>
<th>time</th>
<th>solved</th>
<th>obj</th>
<th>nodes</th>
<th>fails</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:1</td>
<td>100</td>
<td>452</td>
<td>2034</td>
<td>352</td>
<td>42.2</td>
<td>100</td>
<td>381</td>
<td>163</td>
<td>0</td>
<td>3.5</td>
</tr>
<tr>
<td>3:1</td>
<td>94</td>
<td>1264</td>
<td>3119</td>
<td>3645</td>
<td>75.2</td>
<td>100</td>
<td>749</td>
<td>394</td>
<td>0</td>
<td>8.4</td>
</tr>
<tr>
<td>4:1</td>
<td>65</td>
<td>3213</td>
<td>4574</td>
<td>11476</td>
<td>129.3</td>
<td>100</td>
<td>1349</td>
<td>836</td>
<td>0</td>
<td>18.7</td>
</tr>
<tr>
<td>5:1</td>
<td>10</td>
<td>7475</td>
<td>6878</td>
<td>47590</td>
<td>241.2</td>
<td>100</td>
<td>2312</td>
<td>1585</td>
<td>44</td>
<td>37.7</td>
</tr>
<tr>
<td>6:1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>86</td>
<td>4092</td>
<td>2884</td>
<td>2863</td>
<td>71.5</td>
</tr>
</tbody>
</table>

From 2k5 VMs/500 servers to 10kVMs/2k servers

Table 2: Impact of the datacenter size on the solving process (repair mode).

<table>
<thead>
<tr>
<th>Set</th>
<th>#servers</th>
<th>#VMs</th>
<th>solved</th>
<th>obj</th>
<th>nodes</th>
<th>fails</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>500</td>
<td>2,500</td>
<td>100</td>
<td>1160</td>
<td>805</td>
<td>13</td>
<td>7.0</td>
</tr>
<tr>
<td>x2</td>
<td>1,000</td>
<td>5,000</td>
<td>99</td>
<td>2321</td>
<td>1594</td>
<td>17</td>
<td>36.2</td>
</tr>
<tr>
<td>x3</td>
<td>1,500</td>
<td>7,500</td>
<td>99</td>
<td>3476</td>
<td>2374</td>
<td>43</td>
<td>105.5</td>
</tr>
<tr>
<td>x4</td>
<td>2,000</td>
<td>10,000</td>
<td>100</td>
<td>4635</td>
<td>3171</td>
<td>15</td>
<td>217.0</td>
</tr>
</tbody>
</table>
[RR report: “CP 2011” back to work]

In bin packing
- set variables used to iterate only. Overkill
- switch to internal bitsets + channelling with the placement variables

In side constraints
- set variables are overkill as the list of VMs is known
- switch to 2 list of placement variables

Side change in a heuristic
- Fix a bug that ignored the first-fail
- Improve the scheduling heuristic to branch on VMs going to leaf node
Up to a x20 speedup
No more backtracks
At least 1 sol for every instance
(Better score)

Nothing interesting for a publication (at least in my domain)

Before, the model was overkill
2013-2016
looking for a better migration scheduler
[Vincent Kherbache work]

Experiment setup

[execplace vanilla, entropy, cloudsim, ...]

network and workload blind
network and workload aware

2013-2016
looking for a better migration scheduler

btrplace + migration scheduler [UCC 15, TCC 17]

Experiment setup

network and workload aware
Coding effort

Network Model
- heterogeneous network
- 1 cumulative constraint per network element;
  +/- 300 sloc.

Migration Model
- memory and network aware
  +/- 200 sloc.

Constraints Model
- restrict the migration models
  +/- 100 sloc.
Phase 0 (6 mo.)
observe, experiment

Phase 1:
mathematical model
(Few months)

Migration Model

\[ d_{CP}(m) = \frac{CP_s}{bw(m) - CP_r} \]
\[ d_{HP}(m) = \frac{HP_s}{bw(m)} + \frac{HP_s - (D \times bw(m))}{bw(m) - HP_r} \]
\[ d(m) = d_{min}(m) + d_{CP}(m) + d_{HP}(m) + D \]
Phase 2: CSP models

Try 1 - maths to CSP 101
Division, truncation issues
does not scale

Try 2: chunk based bandwidth allocation
\textit{e.g.} : 250, 500, 750, 1Gb

Table constraint
better scalability

Try 3: ... why slowing down migrations?
Force max bandwidth
The duration as a constant
Evaluating the benefits of “max bandwidth”

<table>
<thead>
<tr>
<th>Scale</th>
<th>MaxBandwidth option</th>
<th>disabled</th>
<th>enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>65%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>22%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>83%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x10</td>
<td>62%</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scale</th>
<th>MaxBandwidth option</th>
<th>disabled</th>
<th>enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>66%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>57%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>47%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x8</td>
<td>0%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>x18</td>
<td>0%</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

**Infrastructure size**

- 960 VMs, 20x24 nodes

**Amount of VMs**

- 1728 VMs, 2x24 nodes
Placement + migration scheduler

Back to 2006 solving workflow :D

Phase 1: placement & old schedule
Phase 2: new scheduler (requires the route)

Not sure about the need to consolidate the phase
Can’t saturate the network
Workarounds exist
2017
The story of the knapsack
The right filtering algorithm for the right workload

The graph shows a scatter plot with the x-axis representing CPU load (%) and the y-axis representing solving duration (ms). The data points cluster around the 32 instances mark, with a notable concentration near the 30000 ms timeout boundary. The graph highlights very high load instances and small but hard instances, with comments indicating that it's 'ok when non-solvable but no evidence'.

Key observations:
- **32 instances** are marked near the timeout boundary.
- **Very high load** instances and **small but hard instances** are noted.
- It's 'ok when non-solvable but no evidence'.
knapsack filtering

simple to understand, to develop

“Iterate over the candidate items and filter out the oversized”
called after every assignment

O(n) worst case complexity
dsc. sorting items helps …
but costly with several dimensions

Sort items per dimension inside the constraint
Indirection tables to ensure compatibility with the core model
knapsack filtering

Strong filtering iff packing objective
  high load
  big items

Low filtering when balance objective
  low load
  small items

With sorted item, memory usage increases with the dimensions
  Overhead > benefits
Triggered Knapsack

Triggered if the residual < biggest item larger constants

32 instances

16 instances
Review everything

understand the workload, tune the model, tune the solver, tune the heuristics

38cfc58: [Choco.4 FirstFail] heuristic from the O(n) avg. legacy first fail to a O(n) worst case (iterates from the last instantiated variable, stop when Dom size == 2)

e6a834d: [task scheduling] simplistic local entailment. Stop early if all variables are instantiated

ab32549: [vector packing] cap cardinalities wrt. resource usage

ef57234: [memory] allocate memory per chunk to prevent increase with copy
STOP
RECAP
I am from the (distributed) system community
Not from the CP community
Composability is gold

Fit the way we think, we work, we upgrade
Take care of flexibility overhead
Multi expertise required

Domain expertise
CSP / solver expertise
Engineering expertise
global constraints are effective

But re-usability has its limits
“The right model/filtering for the right problem”

“The granularity is good when going finer does not change the decisions”
— Fabien Hermenier, right now
A regular dev writing a constraint?

Deterministic event-based programming
  Error prone, no one like

Hard to debug
  Jump to the 36th instance of constraint X, at the
  217th search nodes, in the 47th. awakeOnRem

Unusual reasoning
Help me at modeling, developing, checking redundancy checkers, effectiveness checkers, convenient programming model, debugger, ...

Wish list of one CP enthusiast
http://BtrPlace.org

production ready  live demo  stable user API  documented  tutorials
issue tracker  support  chat room