Looking for the perfect VM scheduler

Fabien Hermenier
— placing rectangles since 2006
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

VM scheduling, green computing
Entreprise cloud company
“Going beyond hyperconverged infrastructures”

VM scheduling, resource management
Virtualization
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
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 the same business value with a smaller infrastructure.
KEEP CALM AND CONSOLIDATE AS HELL

1 node =

VDI workload:
12+ vCPU/1 pCPU

100+ VMs / server
static schedulers

consider the VM queue
deployed everywhere \([1,2,3,4]\)
fragmentation issues

dynamic schedulers

live-migrations \([5]\) to address fragmentation

Costly (storage, migration latency)
thousands of articles \([10-13]\)
over-hyped ? \([9]\)

but used in private clouds \([6,7,8]\)
(steady workloads ?)
Placement constraints

various concerns

performance, security, power efficiency, legal agreements, high-availability, fault-tolerance ...

dimension

spatial or temporal

enforcement level

hard or soft

manipulated concepts

state, placement, resource allocation, action schedule, counters, etc.
**discrete constraints**

\[ \text{spread}(\text{VM}[1,2]) \]
\[ \text{ban}(\text{VM}1, \text{N}1) \]
\[ \text{ban}(\text{VM}2, \text{N}2) \]

“simple” spatial problem

**continuous constraints**

\[ \text{spread}(\text{VM}[1,2]) \]
\[ \text{ban}(\text{VM}1, \text{N}1) \]
\[ \text{ban}(\text{VM}2, \text{N}2) \]

harder scheduling problem
(think about actions interleaving)
hard constraints

soft constraints

spread(VM[1..50])

must be satisfied

all or nothing approach

not always meaningful

mostlySpread(VM[1..50], 4, 6)

satisfiable or not

internal or external penalty model

harder to implement/scale

hard to standardise?
High-availability

x-FT VMs must survive to any crash of x nodes

1 - FT

0 - FT

exact approach: solve $n^x$ placement problems [17]
The VMWare DRS way

- slot based
- catch the x-biggest nodes
- checks the remaining free slots
- simple, scalable
- waste with heterogeneous VMs
- cluster based
The constraint catalog evolves

- Dynamic Power Management (DRS 3.1) in 2009?
- VM-VM affinity (DRS) in 2010?
- Dedicated instances (EC2) in mar. 2011
- VM-host affinity (DRS 4.1) in apr. 2011
- MaxVMsPerServer (DRS 5.1) in sep. 2012
- The constraint needed in 2014
- 2016
the objective

provider side

\( \min(x) \) 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?

- \( \min(\ldots) \) or \( \max(\ldots) \)
- Easy to say
- Hardly provable
- Composable through weighting magic

- Threshold based
- Domain specific expertise
- Verifiable
- Composable
Acropolis Dynamic Scheduler\textsuperscript{[18]}  
Hotspot mitigation

**Trigger**: 15 min

**Thresholds**: 85%  
- CPU
- storage-CPU

**Maintain**:  
- affinity constraints

**Minimize**: \( \sum_{\text{mig.}} \text{cost} \)  
- Resource demand  
  (from machine learning)
BtrPlace

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
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)

interaction though a DSL, an API or JSON messages

spread(VM[2..3]);
preserve(VM1,’cpu’, 3);
offline(@N4);
An Open-Source java library for constraint programming

deterministic composition
high-level constraints

\[ \mathcal{X} = \{x_1, x_2, x_3\} \]
\[ \mathcal{D}(x_i) = [0, 2], \forall x_i \in \mathcal{X} \]
\[ \mathcal{C} = \left\{ \begin{array}{c}
    c_1 : x_1 < x_2 \\
    c_2 : x_1 + x_2 \geq 2 \\
    c_3 : x_1 < x_3
  \end{array} \right\} \]

the right model for the right problem
BtrPlace core CSP

models a reconfiguration plan
1 model of transition per element
action durations as constants *

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

\[\begin{align*}
\text{relocatable}(v \in V) & \triangleq \ldots \\
\text{shutdown}(v \in V) & \triangleq \ldots \\
\text{suspend}(v \in V) & \triangleq \ldots \\
\text{resume}(v \in V) & \triangleq \ldots \\
\text{kill}(v \in V) & \triangleq \ldots \\
\text{bootable}(n \in N) & \triangleq \ldots \\
\text{halttable}(n \in N) & \triangleq \ldots
\end{align*}\]
Views bring additional concerns
new variables and relations

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

ShareableResource(r) ::= …

Network() ::= …

Power() ::= …

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

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

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

\[
\ldots
\]
vector packing problem

items with a finite volume to place inside finite bins

generalisation of the bin packing problem

the basic to model the infra.
1 dimension = 1 resource

NP-hard problem
how to support migrations

temporary, resources are used on the source and the destination nodes
Migrations are costly
Dynamic Schedulers

Using Vector packing [10,12]

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

Solution #1: 1m, 1m, 2m
**dynamic schedulers**

Using Vector packing $[10,12]$  

min(#onlineNodes) = 3

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

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

lower MTTR (faster)
dynamic scheduling using vector packing

[10, 12]

offline(N2) + no CPU sharing
Dependency management

1) migrate VM2, migrate VM4, migrate VM5
2) shutdown(N2), migrate VM7
coarse grain staging delay actions

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

stage 1  

stage 2  
time
Resource-Constrained Project Scheduling Problem [14]
Resource-Constrained Project Scheduling Problem

1 resource per (node x dimension), bounded capacity

tasks to model the VM lifecycle.
  height to model a consumption
  width to model a duration

at any moment, the cumulative task consumption on a resource cannot exceed its capacity

comfortable to express continuous optimisation

NP-hard problem
From a theoretical to a practical solution

duration may be longer
convert to an event based schedule

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
Extensibility in practice
looking for a better migration scheduler

Experiment setup

network and workload blind
Extensibility in practice
looking for a better migration scheduler

Experiment setup

network and workload aware
Extensibility in practice
solver-side

Network Model
- heterogeneous network
- cumulative constraints; +/- 300 sloc.

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

Constraints Model
- restrict the migration models
- +/- 100 sloc.
Nobody’s perfect

placement

vector packing problem

scheduling

multi-mode resource-constrained project scheduling problem

exact approaches:

1000 VMs / 10 nodes -> \(10^{1000}\) assignments

heuristics approaches: fast but approximatives

scaling

NP-hard problems
the search heuristic

per objective

guide choco to instantiation of interest at each search node

1. which of the variables to focus
2. which value to try

do not alter the theoretical problem

............[1/2] relocatable(vm#97).cSlice_end = {1}
.............[2/2] relocatable(vm#202).cSlice_end = {2}
............[1/2] relocatable(vm#202).cSlice_end = {4}
............[1/2] relocatable(vm#203).cSlice_end = {2}
manage only supposed mis-placed VMs

beware of under estimations!

scheduler.doRepair(true)

static model analysis 101

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

Partitioning benefits

2013 perf numbers...

//! non Nutanix workloads
Master the problem
understand the workload,
tune the model, tune the solver, tune the heuristics

( benching on my laptop)

//!\ non Nutanix workloads
"current" performance

![Graph showing the relationship between time (in seconds) and virtual machines (x 1,000)]

- **Kind**: li, nr
- **Xenon servers**: non Nutanix workloads
The right filtering algorithm for the right workload

32 instances

very high load
small but hard instances

ok when non-solvable
but no evidence
The costly Knapsack filtering to the rescue

- Smarter but slower
- Higher memory consumption
- Bigger constants

Timeout

16 instances

CPU load (%) vs. solving duration (ms)
RECAP
The VM scheduler makes cloud benefits real
think about what is costly
static scheduling for a peaceful life
dynamic scheduling to cease the day
no holy grail
master the problem
with great power comes great responsibility
http://BtrPlace.org

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

(once graduated)

NUTANIX™
Member of Technical Staff
San Jose, California

2 yrs. postdoc
Sophia, France

resource management in edge computing

Efficiently connecting CLOUD & EDGE
References

1. Omega: flexible, scalable schedulers for large computer clusters. Eurosys’13
2. Sparrow: distributed, low latency scheduling, SOSP’13
3. Large-scale cluster management at Google with Borg. Eurosys 15
4. Firmament: fast, centralized cluster at scale. OSDI 16
5. Live-migration of virtual machines. NSDI’05
6. VMWare DRS. 2006
8. Nutanix Acropolis Dynamic Scheduler. 2017
10. Entropy: a consolidation manager for clusters. VEE 2009
11. pMapper: power and migration cost aware application placement in virtualized systems. Middleware 2009
12. Memory Buddies: exploiting page sharing for smart consolidation in virtualised data centres. VEE 2009
13. Energy-aware resource allocation heuristics for efficient management of data centres for cloud computing. FGCS 2012
14. BtrPlace: a flexible consolidation manager for highly available applications. TDSC 2013
15. Higher SLA satisfaction in datacenter with continuous VM placement constraints. HotDep 2013
16. Scheduling live-migrations for fast, adaptable and energy-efficient relocation operations. UCC 2015
17. Guaranteeing high availability goals for virtual machine placement. ICDCS 2011