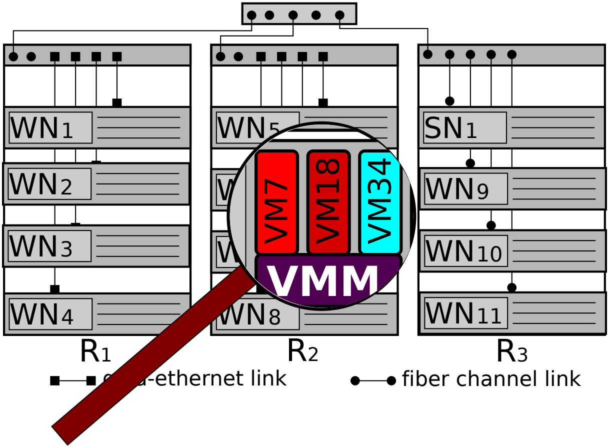
BtrPlace: Autonomous and Flexible Management of VMs in Hosting Platforms

Fabien Hermenier

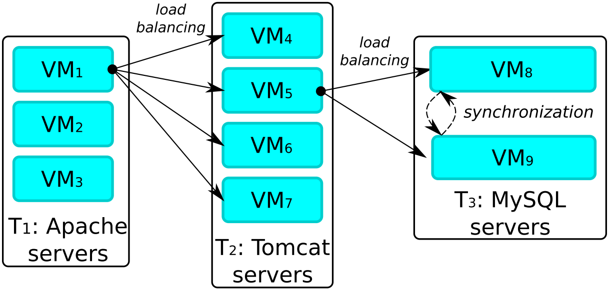
OASIS - Team Project
University of Nice-Sophia Antipolis

March 28, 2012

Virtualized datacenter



Virtualized highly-available Web application



Dynamic consolidation meets high-availability

Datacenter administrator wants

- to stack VMs on servers to improve resource usage
- an autonomous management of the VMs

Application administrator wants its VMs placed wrt. :

- their resource requirements
- fault tolerance to hardware failure for replicated services
- a network latency compatible with the synchronization protocol

One step beyond

Some problems

- multiple specific placement constraints
- concurrent/overlapping constraints
- non-expert users with limited concerns
- new placement constraints emerge with new usages

One proposition

- an extensible, composable VM manager
- specialized on the fly by independent constraints expressed by users
- easy specification of placement constraints with declarative scripts

Sample scripts

Datacenter description

```
namespace datacenter;

$servers = @srv[1 .. 12];
$racks=$servers % 4;

export $racks to *;
```

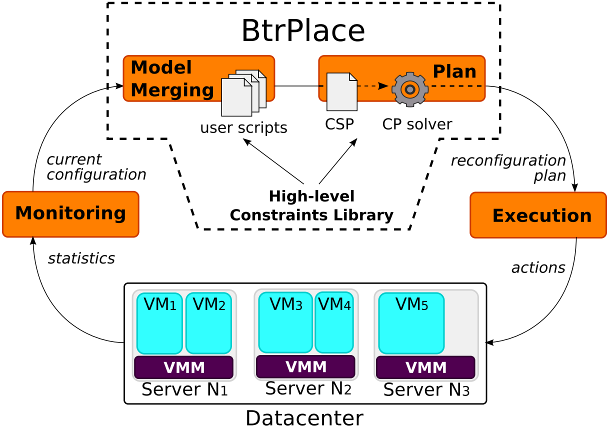
Datacenter requirements

```
namespace sysadmin;
import datacenter;
import clients.*;

vmBtrPlace : large;

fence(vmBtrPlace,@srv1);
lonely(vmBtrPlace);
ban($clients, @srv5);
```

Application description

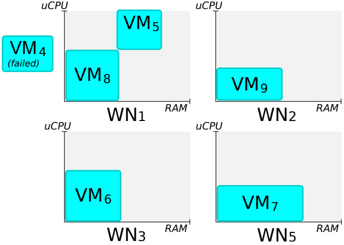
```
namespace clients.app1;

import datacenter;

VM[1..7]: small<clone, boot=5, halt=5>;
VM[8, 10]: large<clone, boot=60, halt=10>;

$T1 = {VM1, VM2, VM3};
$T2 = VM[4..7];
$T3 = VM[8, 10];

for $t in $T[1..3] {
   spread($t);
}
among($T3,$racks);

export $me to sysadmin;
```

Integration into Entropy



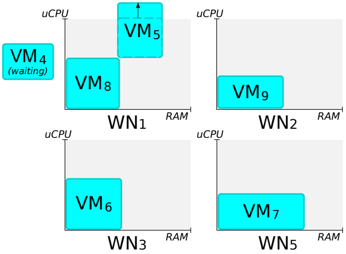
production code: 12k loc
unit tests: 8k loc
API documentation: 5k of javadoc

Sample loop iteration - Monitor



Retrieves the current state of the datacenter

Sample loop iteration - Model merging



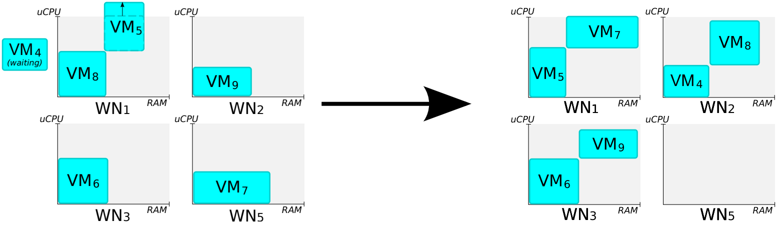
Current configuration is not viable:

- VM_4 must be running
- VM_5 does not have access to sufficient uCPU resource
- WN_5 should not host any VMs

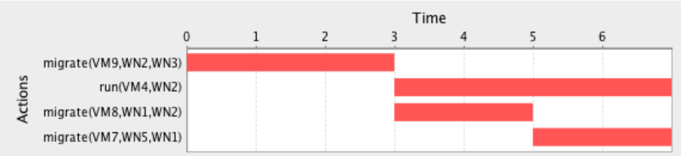
Reconfiguration plan:
actions on VMs and servers to reach a viable configuration
migration, *suspend*, *resume*, *shutdown*, *startup*, . . .

Sample loop iteration - Plan

1. compute a viable placement for the VMs



2. schedule the actions



The reconfiguration problem

Two problems in one

- place the VMs: multi-dimensional packing with restrictions
- schedule the actions: continuous resource restrictions, responsiveness

The approach : constraint programming

- generation of a core model
- placement constraints are translated into "CP constraints" then plugged into the core model

Constraint Programming 101 - Model a problem as a CSP

$$
\begin{aligned}
\mathcal{X} &= \{x_1, x_2, x_3\} \\
\mathcal{D}(x_i) &= [0, 4]\,, \forall x_i \in \mathcal{X} \\
\mathcal{C} &= \begin{cases} c_1 : & x_1 < x_2 \\ c_2 : & x_1 + x_2 + x_3 = 4 \\ c_3 : & \mathit{allDifferent}(x_1, x_2, x_3) \end{cases}
\end{aligned}
$$

- high-level standardized constraints
- good expressivity
- deterministic composition

- hard to develop efficient custom constraints

Constraint Programming 101 - Solving a CSP

Solving algorithm

- generic : DFS customizable by search heuristics, filtering, propagation
- independent from the constraints composing the model

- deterministic solving process
- portability of a model (somewhat)

- exact solving duration
- bad model leads to bad performance

Modeling the core Reconfiguration Problem (RP)

Data from the provisioning module

- VMs : current state, next state, resource consumption
- servers: current state and resource capacities

Inside a reconfiguration : actions

- resource usage distribution changes
- actions are modeled wrt. their impacts on resources

In practice, 5 500 loc; Choco library

Modeling actions using *slices*

Finite period where CPU and memory resources are consumed on a server

- *c-slice*: Resources are currently consumed on a known server
- *d-slice*: Resources will be consuming on a server at the end of the reconfiguration

Each slice exposes using variables :

- its placement
- its resource consumption
- the time interval it consumes resources

Building block to model actions and express constraints

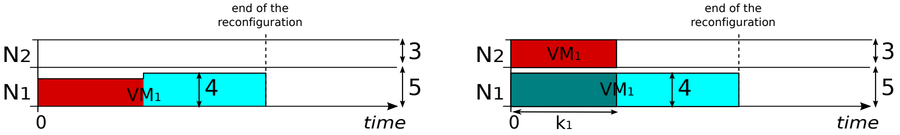
Actions

Each action exposes using variables :

- the moment it starts and terminates
- its cost
- the associated slices

Modeling a *migration*

- VM_1 consumed 3 uCPU : one c-slice
- VM_1 now requires 4 uCPU : one d-slice
- a migration occurs iff. slices are not co-located (est. duration k_1).



(a) VM_1 was running on N_1 (b) VM_1 was running on N_2

Solving the core problem with CP

Modeling the coordination between slices

- no overloaded servers: two 1D *bin-packing* constraints to place d-slices wrt. their resource usage
- actions scheduling: a home-made constraint to manipulate slices (similar to *cumulatives*)

Solving: truncated DFS with custom heuristic

Oriented for responsiveness

- place the d-slices for fewest and cheapest actions
- schedule the d-slices asap

A first library

12 constraints covering

- multiple concerns :
  - resource mgmt: *capacity*, *preserve* , *oversubscription*, *offline*, *noIdles*
  - reliability: *spread*, *root*
  - partitioning: *among*, *ban*, *fence*
  - security: *lonely*, *quarantine*
- multiple aspects of a reconfiguration: VM placement, VM resource allocation, server state, actions schedule, relocation method

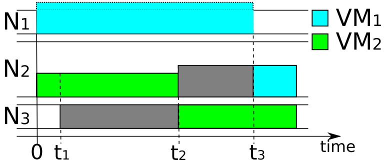
Concise implementation

- about 30 loc. each,
- half a day to implement *lonely* from EC2 specification

Implementation of spread

`spread({VM1, VM2})`

VMs must not overlap on a same server

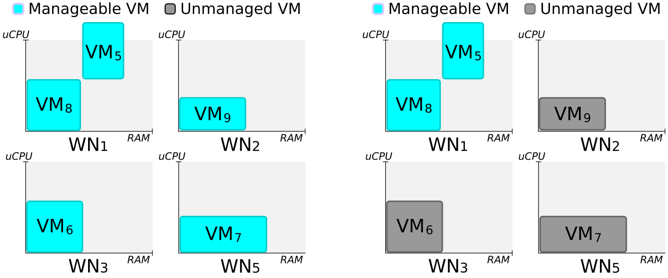


$$\forall V \subseteq \mathcal{V},\ \texttt{spread}(V) \triangleq$$
$$allDifferent(\{d_i^{host} | v_i \in V\})$$
$$implies(eq(d_i^{host}, c_j^{host}),\ geq(d_i^{st}, c_j^{ed})),\ \forall v_i, v_j \in V$$

50 lines of code

Improving the solving process

The *filter* optimization



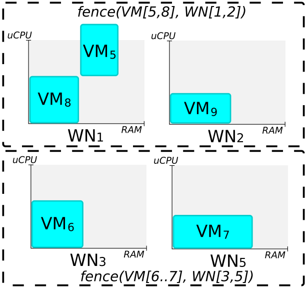
(c) Wo. filter

(d) With filter

- each constraint checks for misplaced VMs
- the CSP is a sub RP as reduced as possible
- beware of false positives

Improving the solving process

Partitioning



- constraints may lead to disjoint sub-RPs
- sub-RPs are solved in parallel
- beware of oversized partitions or un-perfect partitioning

Recovering from external events - RUBiS Benchmark

- 8 servers host 21 VMs running 3 RUBiS benchmarks
- the datacenter administrator uses *ban* constraints to prepare software maintenance

Time	Event	Reconfiguration Plan: Actions	Reconfiguration Plan: Duration
2'10	+ `ban({WN8})`	3 + **3 migrations**	0'42
4'30	+ `ban({WN4})`	2 + **7 migrations**	1'02
7'05	- `ban({WN4})`	no reconfiguration	
11'23	+ `ban({WN4})`	**no solution**	
11'43	- `ban({WN8})` + `ban({WN4})`	**2 migrations**	0'28

- hidden side effects on BtrPlace, not the datacenter administrator

Scalability evaluation

Simulated instances

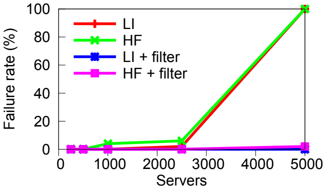
- from 1,000 to 5,000 servers grouped by 250
- 3-tiers Web applications (20 VMs each)
- initial placement and uCPU usage computed pseudo-randomly
- consolidation ratio of 6:1
- global resource usage: 65% memory, 73% uCPU

2 scenarios

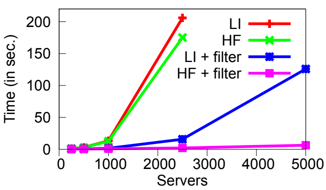
- Hardware Failure (HF): 0.5% of the servers are turned off
- Load Increase (LI): 10% of the applications ask for 30% more resources (+5% overall usage)

Impact of the *filter* optimization

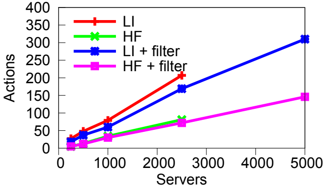
- better scalability : faster solving process
- better reconfiguration plans : smaller and faster



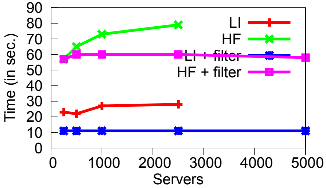
(e) Failure rate



(f) Solving duration



(g) Nb. of actions

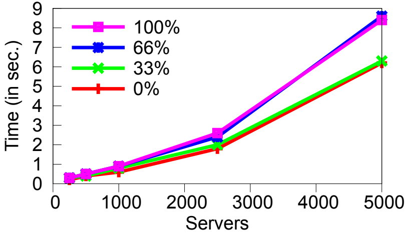


(h) Reconfiguration duration

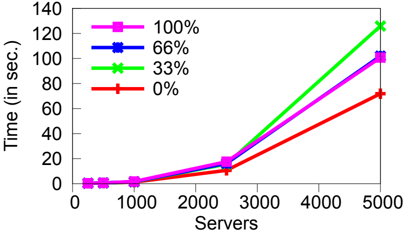
Impact of the placement constraints

A varying ratio of applications have placement constraints
(3 `spread` + 1 `among` each)

Solving duration



(i) HF scenario



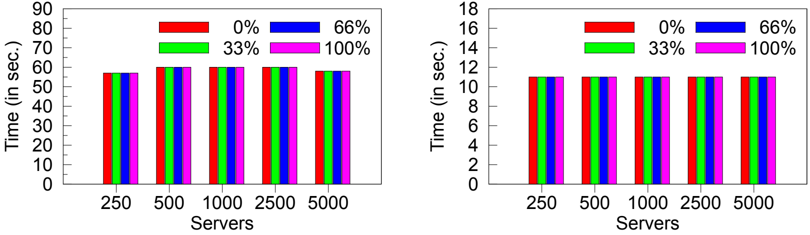
(j) LI scenario

The core RP still dominates the solving process

Impact of the placement constraints

A varying ratio of applications have placement constraints
(3 `spread` + 1 `among` each)

Reconfiguration plans



(k) HF scenario

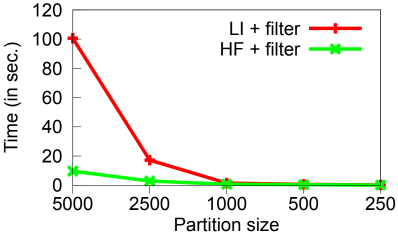
(l) LI scenario

No impact

Impact of partitioning

- 5.000 servers, 30.000 VMs, 1.500 x (3 *spread* + 1 *among* + 1 *fence*)
- Partitions: from 1 x 5.000 servers to 20 x 250 servers.

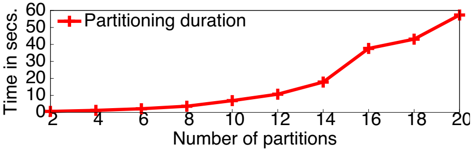
Solving duration



The smaller, the better

Impact of partitioning

Varying number of 2.500 servers partition



14 x 2.500 servers (14 shipping containers):

- 35,000 servers hosting 210,000 VMs
- 20 seconds (partitioning) + 20 seconds (solving duration)

BtrPlace

Flexibility

- a composable reconfiguration algorithm
- manipulable elements: VM placement, VM resource allocation, actions schedule, relocation method, server state
- a first library of 12 concise placement constraints to express dependability requirements
- the Fit4Green FP7:
  - not affiliated to BtrPlace, nor familiar with CP
  - implement a power model and placement constraints
  - no real modifications of the core RP

BtrPlace

Performance

- placement constraints introduce an acceptable overhead
- 5.000 servers hosting 30.000 VMs with 6.000 constraints
  - 120 seconds wo. partitioning
  - 20 seconds with partitions of 2.500 servers
- scalability limited by the partitions size and the number of slaves

Last words

Next BtrPlace

- new concerns : network, storage, ...
- BtrPlace Constraint Catalog
- new manageable elements: VM state, hosting platform
- automatic and optimistic partitioning
- penalty cost