Placement constraints for a better QoS in clouds

Extending BtrPlace to support typing

Mathieu Bivert
Tutor : Fabien Hermenier

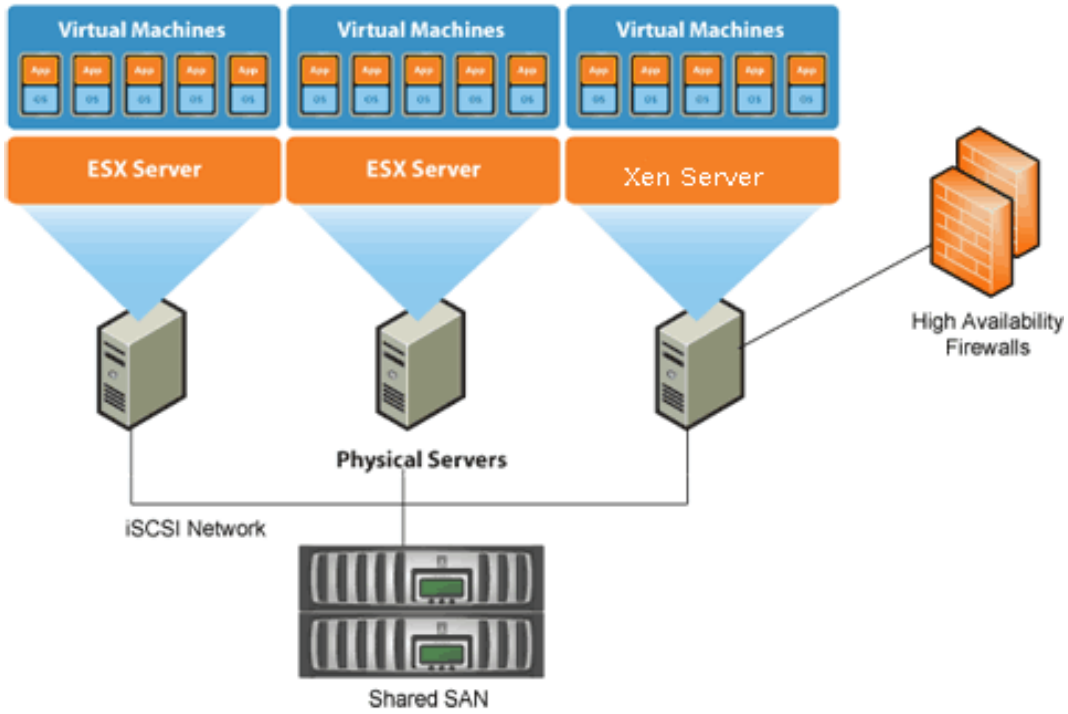
Polytech'Nice Sophia

March 8, 2013

Map

Introduction
Some context
Virtualisation and Cloud
BtrPlace, a placement manager
Adding typing in BtrPlace
More vocabulary
Special case
General case
Additional constraints
Management
Problems encountered
Incomplete work
To sum up

Some context



We define QoS as the performance, the avaibilitity, etc. provided by a cloud. Virtualization in clouds allows to

- Launch and stop services on the fly
- Replicates easily VMs running those services
- Facilitate administration

Clouds in business

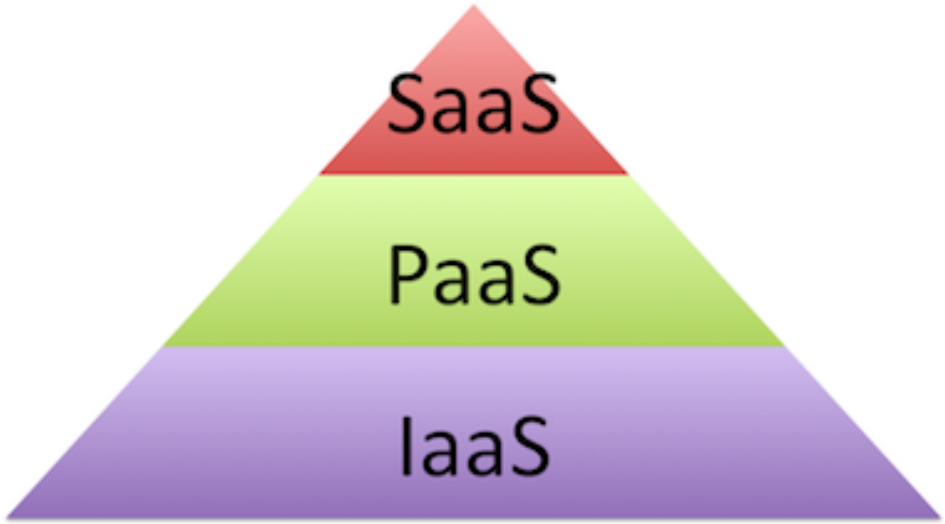
Large firms delegates their IT infrastructure to specialized companies

- Reduction of the costs (less hardware to buy and manage, less software to write, etc.)
- Augmentation of the QoS

However, by doing so, those firms:

- Lose control over their data
- Become dependent of another company

Different types of services



BtrPlace works at the Infrastructure level.

How is it done?

There are different kind of hypervisor,

How is it done?

There are different kind of hypervisor,
with different features,

How is it done?

There are different kind of hypervisor,
with different features,
and different licences;
Need for a software to manage at the Infrastructure level those different hypervisors, and provide services such as redundancy of services.

BtrPlace

BtrPlace is a software written by Fabien Hermenier (OASIS team).

BtrPlace

BtrPlace is a software written by Fabien Hermenier (OASIS team).
It aims to solve the problem of distributing a set of VMs on a set of nodes efficiently, by following some constraints. The latters can be :

- imposed by the hardware, such as available ressources
- given by the user, following his needs (eg. replication of VMs)
- imposed by hypervisors licences

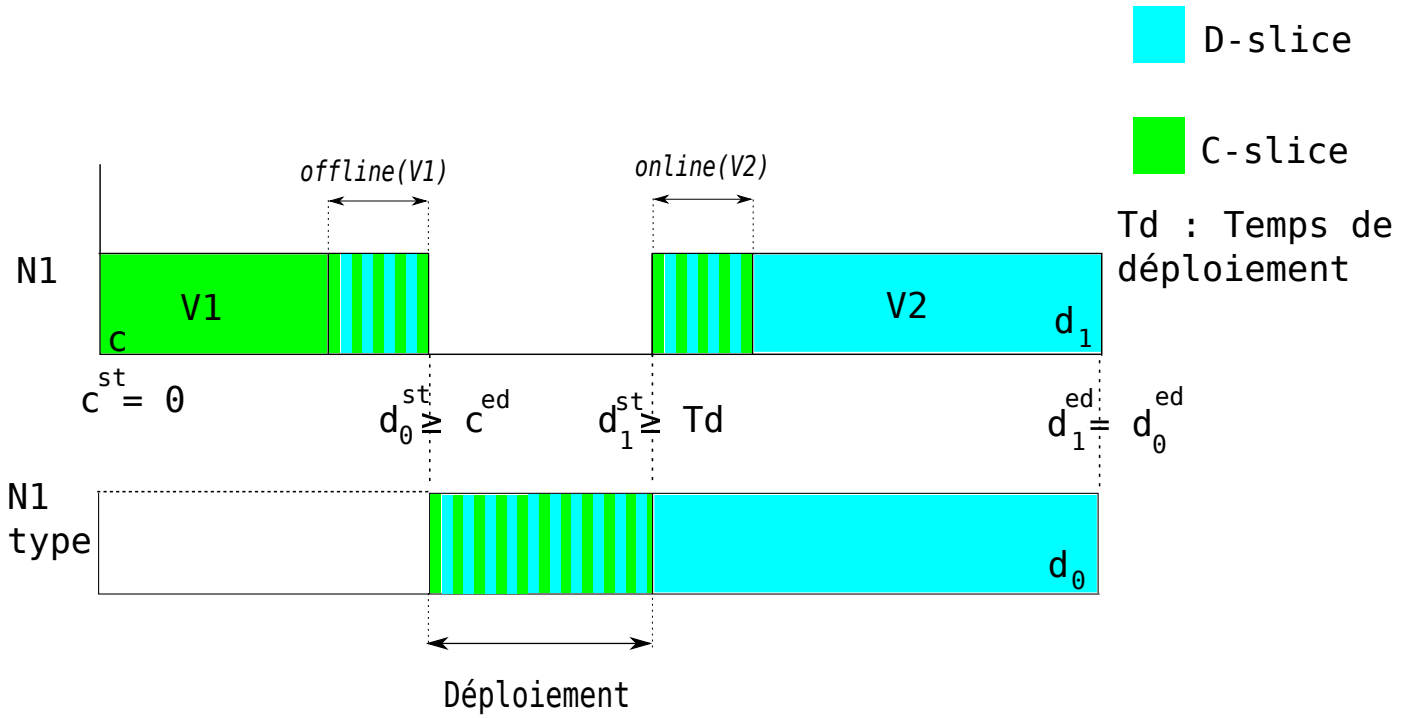
BtrPlace

BtrPlace is a software written by Fabien Hermenier (OASIS team).
It aims to solve the problem of distributing a set of VMs on a set of nodes efficiently, by following some constraints. The latters can be :

- imposed by the hardware, such as available ressources
- given by the user, following his needs (eg. replication of VMs)
- imposed by hypervisors licences

As it competitors, BtrPlace doesn't make the distinction between hypervisor, but as it was designed to be extensible, it should be reasonably easy to augment its model to support typing

More vocabulary



- **Type**, integer associated to each hypervisor
- **Deployment**, operation of rebooting a node and eventually changing its hypervisor
- **Reconfiguration**, operation during which BtrPlace change the placement of VMs on nodes following constraints

Proceeding of the work

We worked incrementally by

1. modeling and implementing a special case of the typing
2. modeling and implementing the general case
3. implementing some constraints associated to typing problems

Model

Hypothesis: we know which nodes are going to change their hypervisor, and the name of the new hypervisor.

Model

Hypothesis: we know which nodes are going to change their hypervisor, and the name of the new hypervisor.
For such a node, the following constraints must be satisfied:

$$P(c) = n \Rightarrow c^{\text{ed}} \leq D^{\text{st}}$$

$$P(d) = n \Rightarrow d^{\text{st}} \geq D^{\text{ed}}$$

Model

Hypothesis: we know which nodes are going to change their hypervisor, and the name of the new hypervisor.
For such a node, the following constraints must be satisfied:

$$P(c) = n \Rightarrow c^{\text{ed}} \leq D^{\text{st}}$$

$$P(d) = n \Rightarrow d^{\text{st}} \geq D^{\text{ed}}$$

Placement satisfied iff:

$$P(v) = n \Rightarrow T(n) = T(v)$$

Code

This special case is implemented through a constraint $Platform((n_i, h_j), (n_{i+1}, h_k), \ldots)$. There are two main methods in this class:

1. **inject**, which inject into Choco the two previous constraints
2. **isSatisfied**, which ensures the injected constraints are indeed satisfied in the new configuration

Model

We delete the previous hypothesis : BtrPlace should deduce the new type of the nodes.
We add a vector v_i to each node. $v_i[t]$ represents the number of VMs running under the hypervisor t.

Model

We delete the previous hypothesis : BtrPlace should deduce the new type of the nodes.
We add a vector v_i to each node. $v_i[t]$ represents the number of VMs running under the hypervisor t. The placement is satisfied iff:

$$(\exists! x \in v_i), x \neq 0$$

Currently, only the model has been defined correctly, no working code.

MinPlatform

MinPlatform(*nodes*, *type*, *n*) ensures at least *n* nodes from *nodes* runs hypervisor *type*.

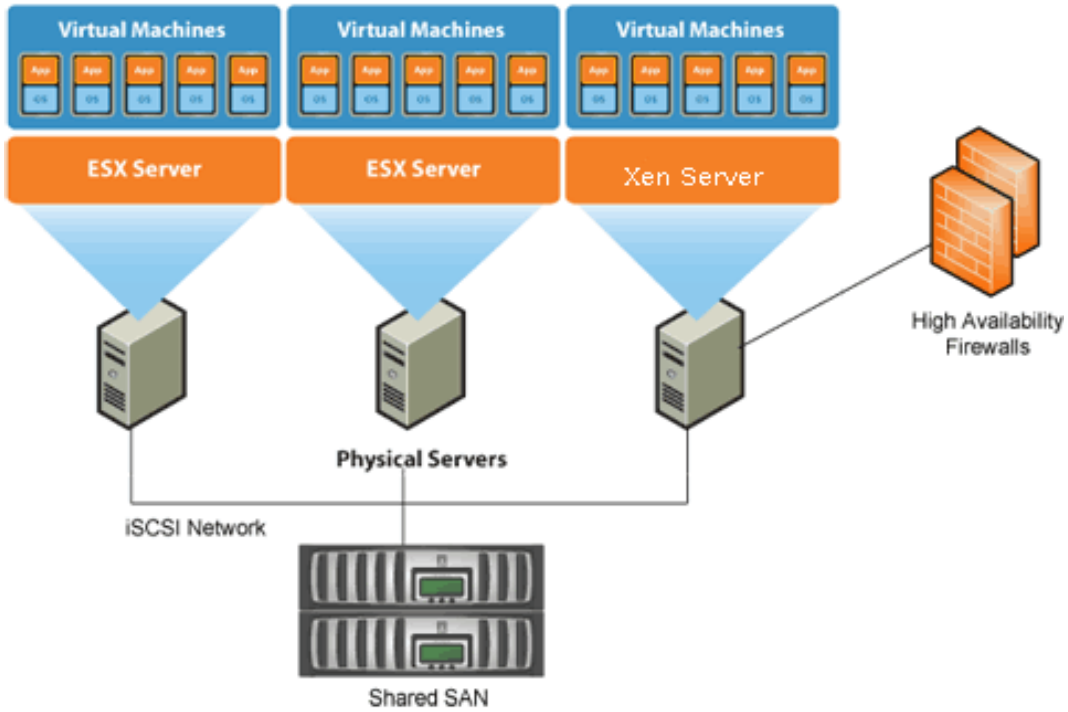


Figure : Here, BtrPlace could ensure at least two nodes run ESX Server

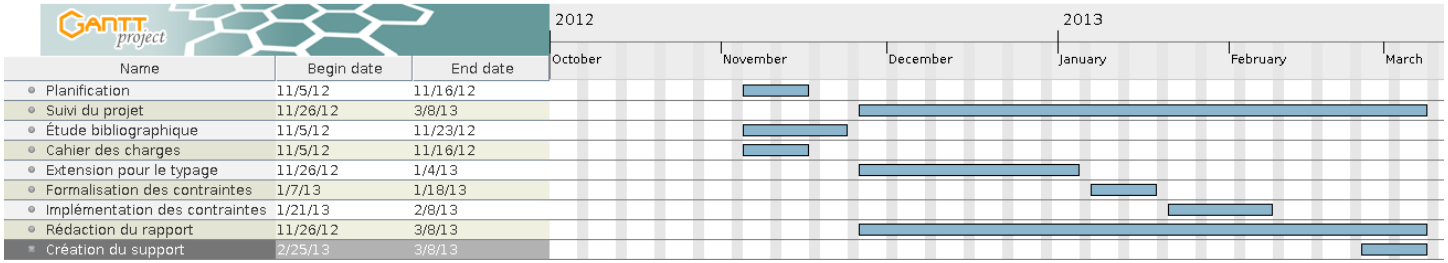
MaxVM

MaxVM(*nodes*, *type*, *n*) ensures at most *n* platforms runs of nodes running hypervisor *type*.

MaxVM

MaxVM(*nodes*, *type*, *n*) ensures at most *n* platforms runs of nodes running hypervisor *type*.
Other PFE projects proposed by Fabien in response to licence limitations (VMWare notably), easily implemented because typing is done with integer.

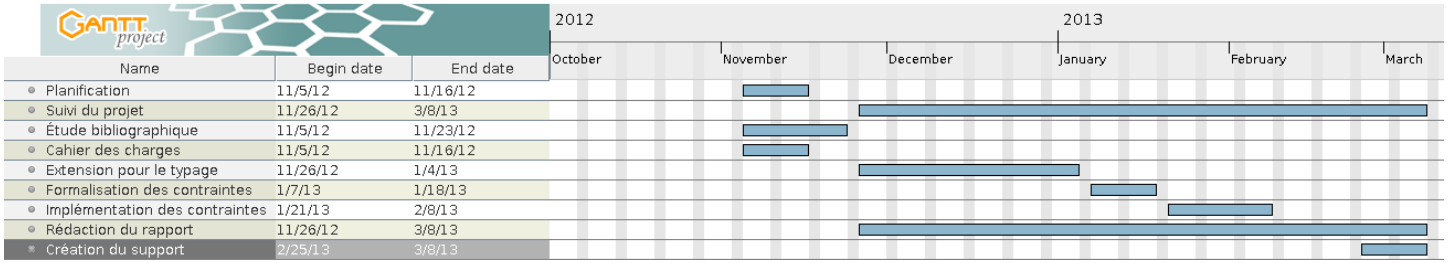
Timing management



Name	Begin date	End date
Planification	11/5/12	11/16/12
Suivi du projet	11/26/12	3/8/13
Étude bibliographique	11/5/12	11/23/12
Cahier des charges	11/5/12	11/16/12
Extension pour le typage	11/26/12	1/4/13
Formalisation des contraintes	1/7/13	1/18/13
Implémentation des contraintes	1/21/13	2/8/13
Rédaction du rapport	11/26/12	3/8/13
Création du support	2/25/13	3/8/13

Problem: not enough time spent on timing the work; incoherence in the DOW observed at the end of the project, leading to badly structured reports.

Timing management



Name	Begin date	End date
Planification	11/5/12	11/16/12
Suivi du projet	11/26/12	3/8/13
Étude bibliographique	11/5/12	11/23/12
Cahier des charges	11/5/12	11/16/12
Extension pour le typage	11/26/12	1/4/13
Formalisation des contraintes	1/7/13	1/18/13
Implémentation des contraintes	1/21/13	2/8/13
Rédaction du rapport	11/26/12	3/8/13
Création du support	2/25/13	3/8/13

Problem: not enough time spent on timing the work; incoherence in the DOW observed at the end of the project, leading to badly structured reports.

Possible Solution : spend more time on timing and structuring the work rather than on writing others parts of the DOW. Try to evaluate better exogenous elements (mainly other scholar works).

Complexity of BtrPlace

Problem : only documentation available : API Java.
Inadequate and insufficient to understand fully BtrPlace.

Complexity of BtrPlace

Problem : only documentation available : API Java.
Inadequate and insufficient to understand fully BtrPlace.
Possible solution : add two more layers of documentation.

1. one describing the general structure of the software, with some example
2. one more precise than the latter, containing information about model generation and how to write simple constraints

What's done and what's missing?

Goals	State
Modelisation of the special case	done
Implementation of the special case	partial (insufficient testing)
Modelisation of the general case	done
Implementation of the general case	partial (not at the right place)
Modelisation of new constraints	not done (easy and fast to do)
Implementation of new constraints	mostly done (lack of printer/getters, testing)

New competences and technologies

During this project, I learnt and revisited:

- Java and related tools (maven, IntelliJ, unit testing)
- Management of ressources and combinatorial problems
- Choco framework
- Git

How to improve what has been done

To be complete, one may add a few more test for the special case.

How to improve what has been done

To be complete, one may add a few more test for the special case.
The general case, should be implemented at the right place, and not as a user constraint, and tested correctly.

How to improve what has been done

To be complete, one may add a few more test for the special case.
The general case, should be implemented at the right place, and not as a user constraint, and tested correctly.
Models for the constraints *MinPlatform* and *MaxVM* should be defined, and implementations revisited following the established models.

Possible evolutions thanks to typing

Hypervisors licences and features can be pretty differents:

- some allows migrating VMs, some don't
- some put restrictions on usable hardware (number of NIC, RAM, CPU usable by the hypervisor)
- etc.

Typing could help modelize those limitations.

Questions

Thanks for your attention and time. Do you have any questions?