btrScript : a safe management system for virtualized data center

Jean-Marc Menaud, Rémy Pottier

btrScript : a safe management system for virtualized data center

Rémy Pottier, Jean-Marc Menaud
École des Mines de Nantes, Ascola (EMN/INRIA,LINA)
Nantes, France
Email: first-name.last-name@mines-nantes.fr

***Abstract*—Virtual machine management in data centers is more and more complex and this is due to the increasing total number of virtual machines. Virtual machine resources and scheduled policies (e.g., consolidation) define the virtual machine placement. This placement is difficult to compute for large infrastructures. Administrators maintain a correct placement by performing actions (e.g., migrate virtual machines, power off servers...) and some time using autonomic schedulers. We propose btrScript: a safe autonomic system for virtual machine management that includes actions and placement rules. Actions are imperative operations to reconfigure the data center and declarative rules specify the virtual machine placement. Administrators schedule both actions and rules, to manage their data center(s). They can also interact with the btrScript system in order to monitor the data center and compute the correct virtual machine placement.**

***Keywords*-cloud computing, virtualization, management, language;**

I. Introduction

With the emergence of cloud computing, the hosting capacity of the data centers has been continuously growing to support the non-stop increasing clients demand. Currently, *Amazon Web Service*[1] adds each day enough capacity to support the whole *Amazon.com* infrastructure as it was during its five first years[2].

Managing a data center implies to regularly manipulate both the VMs and the servers. Common operations include snapshotting, starting, stopping, or resetting of VMs [1], but also starting, halting, rebooting or performing maintenance operations on servers. Each hosted VM has however specific expectations regarding its quality of service and each action must be executed in accordance to its requirements. Typically it is expected to have a sufficient amount of resources to run the VM at peak level, but also a placement that may be compatible with fault tolerance or networking requirements. Finally, its availability may be required at given periods (*e.g.* business hours for remote desktops).

Infrastructure As A Service (IaaS) solutions such as OpenStack[3] or VCenter[4] extremely simplify creations and deployments of virtual environments. However, the management of the VMs concepts did not follow these changes. Virtualized infrastructure management is still relying on manual changes on the environment as well as a reaction to problems after they occur. Such an approach is no longer compatible with an infrastructure composed of thousands of VMs as a system administrator cannot manipulate a large set of VMs insuring that his actions are compatible with the expected quality of service at a given time, but also will be compatible in the future. This situation has led to some new approaches for the infrastructure management employing automation to replace the traditional manual approach. For example, VMWare DRS [2] can react to a load spike and dynamically adapt the VMs placement following a set of rules given by the administrator.

Close to the VMWare DRS functionality, we have worked with and on an autonomic system called Entropy [3], [4]. The term autonomic computing was first introduced by IBM in 2001 to describe self-managing computing systems [5]. To achieve autonomic computing, an architectural view called the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop, has been suggested in [6]. Mainly, an autonomic system is a software component, configured by human administrators using high-level goals. It uses monitored elements (M), internal knowledge (K) of the system to analyse (A), plan (P) and execute (E) tasks based on these high-level goals. The low-level actions to achieve these goals are automatically calculated and executed.

Entropy implements a classical MAPE-K loop [6], and it specially focuses on the planning part (P). Based on constraint programming, the main Entropy's high-level goal is to ensure that placement rules are constantly satisfied, both on system rules (CPU, RAM) and for the administrative rules. From a given current configuration (initial VM placement and rules) Entropy proposes at each loop a new configuration and its associated ordered operations called the reconfiguration plan. Essentially based on VM live migration, the reconfiguration plan allows to switch from the current to the new configuration. The main interest to use an autonomic system like VMWare DRS or Entropy, is that administrators define high-level goals by specifying criteria that characterise desirable states, but leave to the system the task of finding how to achieve that state.

The main drawback of these systems and that they react

[1] http://aws.amazon.com/
[2] http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_Sigmod2011Keynote.pdf
[3] http://openstack.org
[4] http://www.vmware.com/products/vcenter-server/

after a problem occurs. Thus, daily maintenance operations realised by administrators, like VM migration or creation, are verified by the system after completion, at the next MAPE-K execution loop. Therefore, placement rules can be unsatisfied for a time, which may cause degradation of the quality of service.

From our initial research on a Domain Specific Language (DSL) on VM bulk management [7], we propose in this paper btrScript, a safe management system for virtualized infrastructures. BtrScript checks, according to active rules, the validity of all actions and rules performed by administrators. If an action or a rule is invalid, btrScript detects conflicts and displays all active rules involved for each conflict. To resolve a conflict, btrScript proposes a combination of two modules. The first one is a rule management system to modify, suspend or activate rules. The other one, by interacting with Entropy, proposes (if possible) to the administrator a reconfiguration plan that satisfies all rules. In addition, btrScript has an action scheduler that allows specifying time-based operations and rules. Each interactive or scheduling operation is ensured to be compatible with the current but also the future active rules. Finally, we extend the language proposed in [7] to allow administrators an easy placement rules management.

We evaluate btrScript by comparing our rule management with the scripting tool, vSphere PowerCLI, which can be used to manage rules in a VMWare infrastructure.

This paper is structured as follows. The next section presents the new modules of the btrScript architecture. Section III reminds the VMScript syntax then introduces the language extension. Section IV describes mechanisms to ensure the consistency in the infrastructure. Section V explains how to compute reconfigurations that solve infrastructure issues. Section VI details the comparison of the rule management with btrScript and with the vSphere PowerCLI. Section VII summarizes the former research in the field. Finally, section VIII concludes and presents future works.

II. System overview

In virtualized infrastructures, the number of VMs and servers, as well as the resource utilization of the VMs, evolves. Administrators have to regularly re-organize the infrastructure to optimize the resource usage. Moreover specific requirements as fault tolerant mechanisms have to be defined by placement rules.

BtrScript uses mandatory rules existing in Entropy and in the vmWare DRS, the most widely adopted VM scheduler. Actually, rules define by administrators can restrict the VM placement (for example, enforce a VM to stay in the same server). As a rule does not describe actions, when administrators enable rules, no infrastructure reconfiguration is performed even if the rule is broken. The broken rule detection requires the following data:

- static resources: the cpu and memory capacities of VMs and servers ;
- dynamic resources: the cpu and memory utilization of VMs and servers ;
- the VM placement.

The placement rule management implies additional modules to help administrators to maintain the rule consistency and, consequently, the consistency of the infrastructure. So the VMScript extension, btrScript, has three additional modules: a *language extension*, a *guardian module* and a *placement module*.

Administrators interact with btrScript modules through the text console. This console is an interpreter for the VMScript language and its extension that manages placement rules. This *language extension* is described in Section III-C1.

Scheduling VMs in a large infrastructure is a complex problem [8]. Furthermore, some issues in the infrastructure do not require to be fixed because they are temporary (for example, a cpu consumption peak). The *guardian module* periodically analyzes the infrastructure to report violated rules and overloaded servers. In our opinion, a server is overloaded when the sum of the hosted VM cpu consumption equals the cpu capacity of the server. The VM memory is not considered because we assume memory overcommitment is not used and, consequently, the memory utilization of VMs can not be greater than the host memory. This module warns administrators about issues but it does not try to solve them. Administrators choose the appropriate moment to solve issues manually (by executing actions) or by means of the *placement module*.

The *placement module* is an algorithm that computes actions to execute in order to solve infrastructure issues. It has to integrate all the VMScript placement rules to solve them. Entropy is connected to btrScript to use it as the *placement module*.

III. btrScript Language

The VMScript language allows administrators to introspect and reconfigure the virtualized infrastructure. The VMScript extension, btrScript, reuses these low-level operations and adds the placement rule management. Before the extension language presentation (Section III-C1), Section III-A briefly reminds the VMScript syntax to select, introspect and reconfigure elements in the infrastructure.

A. VMScript background

VMScript operations are based on set manipulation in order to deal with a large amount of VMs and servers. This part describes the selection of elements in the infrastructure to, afterwards, perform introspection and reconfigurations.

1) Selection: The architecture of virtualized infrastructures is a compound of two views, a physical view and a virtual one. For example, the physical view can be clusters that includes servers, and, the virtual view can be virtual

jobs (vjobs) that includes VMs. These views define language types and their hierarchical organization. The hosting relation between servers and VMs maps both of the views.

Each element of the infrastructure is typed and has a unique name and some properties to enhance the infrastructure with additional information such as resource consumptions, operating systems (OS), IP addresses, states, etc.

VMScript uses names to get elements. The "[]" operator allows the selection of a set of elements using a sequence of consecutive numbers, or an enumeration. The following expression selects the elements named *pm1*, *pm2*, *pm3* and *pm-master*:

```
pm[1-3,-master]
```

The "[]" operator can also make an union, a difference or an intersection of sets. The selection of the five servers named *pm1*, *pm2*, *pm3*, *hostA* and *hostB* is written by:

```
[ pm[1-3] , host[A,B] ]
```

The binary "/" operator allows to select elements from their type. The first parameter is a set defined with the previous syntax and the second parameter is a type. If the first element is omitted, all elements of the infrastructure with the specific type are selected. For example, the selection of servers of the infrastructure and the selection of VMs of the *cl1* cluster is, respectively, written by:

```
/server
cl1/vm
```

Element properties added to the model can be used to refine a selection thanks to the "{}" operator. For example, get all servers with a *Linux* OS:

```
/server{os == Linux}
```

B. Introspection and reconfigurations

Once elements to manipulate are selected, the operator ":" allows to introspect and reconfigure the infrastructure.

The introspection is available by getting the value of a property. The resource utilization is one of the default property of servers and VMs, so that it is easy to display the cpu consumption of VMs running on the server *s1*:

```
s1/vm:cpu_cons
```

VMScript actions perform infrastructure reconfigurations such as starting or migrating VMs. From a selection, actions can be applied on VMs and servers. For example, start servers and then migrate 3 VMs on one of them is written by:

```
s[2,6]:start
vm[1-3]:migrate s2
```

Supported actions for servers and VMs in the VMScript language are *start*, *stop*, *suspend*, *resume*, *reboot*. The supported action for servers are *createvm* and supported actions for VMs are *snapshot*, *migrate*.

C. The language extension

1) The rule management: Usually, administrators use imperative actions like creating, migrating, stopping VMs in their data center. However, some specific requirements can not be defined in a imperative way because they describe an infrastructure configuration that lasts for a certain period (i.e., an invariant). Imperative actions describe reconfiguration operations to perform whereas administrators want to describe a configuration without defining how to obtain it.

In the VMScript extension *btrScript*, declarative rules aim to specify VM placement and states with rules.

Placement rules implement in the btrScript language are a subset of the constraints used by Entropy [4]. The btrScript language supports the following rules:

- the *group* rule keeps the VMs on one physical server, for example, to optimize network connections. The following rule makes sure the VMs named *vm1*, *vm3* and *myvm* stay on the same server:

  ```
  strongConnection: group [vm[1,3], myvm]
  ```

- the *spread* rule avoids the VMs to run on a same server (e.g., in a fault tolerance context). The following rule makes sure the VMs named *vm1*, *vm3* and *myvm* run on three servers:

  ```
  dbReplica: spread [vm[1,3], myvm]
  ```

- the *on* rule forces a VM to run on specific servers. Administrators can restrict where VMs run to enforce the use of a specific hardware. The following rule makes sure the VM named *vm1* runs on the server *hostA* or on the server *hostB*:

  ```
  vm1Host: vm1 on host[A,B]
  ```

- the *!on* rule is the negation of the *on* rule. It avoids a VM to run on specific servers. The use of *on* or *!on* operators rely on sets to describe. The smaller the set is, the easier is its description. The following rule makes sure the VM named *vm1* runs on a server different from *hostA* or *hostB*:

  ```
  vm1Blacklist: vm1 !on host[A,B]
  ```

- the *run* rule forces VMs to run. It avoids to unfortunately stop or suspend VMs. The following rule makes sure VMs named *vm1* and *vm2* are in a *running* state.

  ```
  alive: run vm[1,2]
  ```

These rules define more accurately the VM placement. As the data center architecture evolves (e.g., VMs are created, cpu consumptions fluctuate...), rules can be added, enabled, disabled or deleted at run-time.

The previous rule declarations show rules that are defined and enabled at the same time. However, administrators can enable and disable rules as they wish, respectively, from the language operators *enable* and *disable*. In the below

example, the *rules* property lists rules. Afterwards, rules are managed from their identifier.

```
blacklist: vm[1-100] !on host[A,B]

vm1:rules
<blacklist> vm[1-100] !on host[A,B] ON

blacklist:disable

vm1:rules
<blacklist> vm[1-100] !on host[A,B] OFF

blacklist:del

vm1:rules
```

2) Timed actions and rules: Reconfigurations in the infrastructure can occur at well-known dates such as the end of a development project or outside business hours. This language extension proposes to schedule actions, rule activations and rule inactivations. The date definition is close to the crontab syntax, with one additional parameter specifying the year of the execution.

Thanks to the date syntax, repetitive tasks such as VM reboot can be scheduled. The following operations stop VMs of the *myproject* vjob during the night every day:

```
[00:21:*:*:*:*] myproject/vm:stop
[00:8:*:*:*:*] myproject/vm:start
```

For a rule, an activation date and/or an inactivation date allow to automate the rule management. The first following declaration inserts the rule and wait the June 3rd 2011 at ten am to enable it permanently. The second one inserts the rule, wait the June 3rd 2011 to enable it and disable it the June 6th 2011 at six pm.

```
[00:10:3:6:*:11] vm1_2: group [@vm1,@vm2]
[00:10:3:6:*:11-00:18:6:6:*:11] vm1_2: group [@vm1,@vm2]
```

IV. Ensuring the infrastructure consistency

Administrators manage virtualized infrastructures with placement rules and actions. Nevertheless, rule insertions must be checked to avoid inconsistencies that imply no correct VM placement exists.

Once rules are inserted, administrators need to know which rules are unsatisfied. The guardian module allows to warn administrators about infrastructure issues as broken rules.

Finally, infrastructure reconfigurations can not violate placement rules. And so, a rule verification occurs by simulating action effects on the model before each action execution.

A. Rule insertion verification

Administrators describe rules from the btrScript language. Before the insertion of a rule in the system, the placement module checks if this rule does not conflict with existing rules. A conflict leads to a resource organization with no viable placement possible. For example, one *on* rule enforcing 3 VMs, each one having 1 Gb of memory, to run on a server with 2 Gb of memory can never be satisfied. The conflict detection only considers static resources because dynamic resources quickly evolves and can not be predicted. Conflicts may also appear with the combination of several rules. The following example illustrates such a situation in an infrastructure composed of 3 servers (*pm1*, *pm2* ,and *pm3*):

```
together: group vm[1,2]
vm1Ban: vm1 !on pm[1-2]
vm2Ban: vm2 !on pm3
```

vm1 and *vm2* must be hosted on the same server. *vm1* must neither be hosted on *pm1* nor *pm2*, and *vm2* must not be hosted to *pm3*. This last rule produces a conflict as the VMs can not be hosted on any server because the group [*vm1*, *vm2*] is excluded from all servers [*pm1*, *pm2*, *pm3*].

The verification of *group* and *spread* rules ensures there is no intersection between *group* and *spread* VMs. For example, 2 placement rules define VM groups. The first one is composed by *vm1*, *vm2* and the second one by *vm1*, *vm3*. These 2 rules involves *vm1*, *vm2* and *vm3* must belong to the same server. So a *spread* rule with *vm2* and *vm3* can not be inserted.

The verification of *on* and *noton* rules ensures each VM or group can be hosted by, at least, one server. In the previous example, all *on* or *noton* rules about *vm1* modify the *vm2* and *vm3* placement. Consequently, the verification ensures at least one server can host the group *vm1*, *vm2* and *vm3*.

Moreover, server sets defined in both a *on* rule and a *noton* rule for a same VM must have an empty intersection. For example, the following rules are not correct:

```
vm1Host: vm1 on pm[1-3]
vm1Ban: vm1 !on pm2
```

Indeed, the server *pm2* is included in both rules, consequently, that does not represent what the administrator wants. This ambiguous declaration is not allowed.

When the rule module detects a rule insertion will lead to a conflict, the rule is not inserted and administrators receive a warning with rules that conflicts. Administrators can disable these rules to insert the new one.

When a scheduled rule is inserted, the rule module computes its activation period. It selects all the active rules that will be enabled during this period. If the rule to insert leads to a conflict, it is not inserted.

The rule module ensures the consistency of the set of rules but it does not check if a rule is true or false. So broken (i.e., false) rules can be inserted and administrators have to fix them in the future.

B. The infrastructure monitoring

In large infrastructures, administrators need information about the infrastructure architecture to maintain it. Monitoring systems like Ganglia [9] allow to collect lots of data from such an architecture in an efficient way. However, the

large amount of VMs and servers and the number of metrics to observe is too huge to be analyzed by an administrator. In btrScript, a guardian module is proposed to analyze monitoring data and warn the administrator when it detects an overloaded server or a broken rule.

The guardian module periodically analyzes monitoring data. As rules can be added even if they are false, the guardian module periodically checks all activated rules and sends warnings to administrators when a broken rule is detected. Afterwards administrators can solve manually unsatisfied rules, remove them (e.g., for out-of-date rules) or use the placement module to solve the placement issues automatically.

C. Actions and placement rules

Administrators perform and schedule actions to reconfigure the infrastructure. As rules restrict the VM placement, when an imperative action is invoked or planned, rules are checked. This verification occurs by:

- selecting satisfied rules including servers and VMs involved in the action at the action execution date. If rules are not satisfied before the action execution, they are not included in the action verification ;
- simulating the action on the model ;
- checking selected rules.

If the action is not compatible with one of the selected rules, it is not executed.

V. The placement module

When overloaded servers (i.e., a server with a cpu consumption equals to 100%) or violated rules occur in the data center, administrators have to fix them. However, finding a VM placement considering rules and dynamic resources is a tedious task for hundreds or thousands of VMs and servers [8]. So btrScript is linked to a placement module in order to compute a list of actions required to obtain a right placement with respect to a scheduling policy, the placement rules and the physical and VM resources. The policy defines how to do the mapping between VMs and servers. Common policies for VM scheduling are load balancing and consolidation. The load balancing policy [10] distributes the cpu load uniformly across the servers whereas the consolidation policy [11] reduces the number of servers which host VMs. Placement rules customize the policy with specific needs described by administrators. The placement module is used for administrators to solve issues in the data center. In our case, the placement module is Entropy [3], a VM scheduler based on constraint programming. As btrScript does not implement its own placement algorithms, the choice of the placement module define the policy range. Entropy provides a *checker* policy, which satisfies cpu and memory requirements of VMs, and a *consolidation* policy.

Administrators invoke the placement module so as to solve issues (broken rules and overloaded servers) in the infrastructure. During this invocation, the placement module builds a problem that includes:

- a set of servers to analyze ;
- a set of VMs corresponding to VMs that run on the server set ;
- a set of rules to apply on the two previous sets.

From the btrScript model, the placement module obviously gets the actual VM placement and the resource usage required to solve the problem. Afterwards this module computes a plan, that is to say a list of actions, and executes it. In the following example, the placement module solves issues in all the data center.

```
/server:solve checker
```

Due to the *run* rule, it is easy to start or resume VMs without specifying any server. For example, the VM *vm1* is in the *off* state. The administrator can start it with the placement module:

```
vm1r: run vm1
/server:solve checker
```

If the administrator wants to run the VM on servers *pm1* or *hostA*, he adds a *on* rule:

```
vm1r: run vm1
vm1On: vm1 on [ pm1, hostA ]
/server:solve checker
```

The application of one policy on all servers of the infrastructure is not always relevant. Administrators may want to consolidate VMs on a cluster and use load balancing on another one. Moreover a huge problem is longer to solve than a small one. A plan for a small problem, that is to say 50 servers and 200 VMs, are solved in few seconds. For a larger problem (1000 servers and 5000 VMs), the solving time is few minutes [12].

Nevertheless, building a problem from a subset of server infrastructure is complex because the set of servers implies the set of rules added to the problem. Therefore the problem defined is smaller but some rules are cut for being integrated and so these rules are partially solved in the infrastructure context. As an example, a data center is composed by 3 servers (*pm1*, *pm2* and *pm3*) and 2 VMs (*vm1* and *vm2*). *vm1* runs on *pm1* and *vm2* runs on *pm2*. The administrator only adds one rule that enforces the VMs *vm1* and *vm2* to belong to the server *pm3*. As the rule is unsatisfied, the administrator decides to fix it by using the placement module. He invokes the placement module on *pm2* and *pm3* servers after adding the rule:

```
vm1On: vm[1,2] on pm3
pm[2,3]:solve checker
```

Consequently, the placement module considers servers *pm2* and *pm3*. The associated set of VMs only contains the VM *vm2* because *vm1* does not run on the server set. The rule is therefore cut and modified to the following rule:

```
vm2On: vm2 on pm3
```

So the placement module executes a migration of *vm2* to *pm3*. The rule is still violated because *pm3* does not host *vm1*.

Rule modifications are mandatory to avoid side effects and transform a small problem to a big one. From the previous example, the hypothesis of the insertion of the whole rule in order to fix the *vm1* placement involves to add *vm1* and its host *pm1*. Now the placement module needs all VMs running on *pm1* and their rules to compute the plan. The rules of these VMs can add other servers and VMs and so on. At the end, the problem to solve can include all servers and VMs of the infrastructure.

VI. Evaluation

In this section, we present a comparison between the VMware vSphere PowerCLI[5] and the btrScript language. Only few solutions can perform operations and add placement rules to virtualized infrastructures. The VMware vSphere solution includes the most popular distributed resource scheduler (DRS) that enables dynamic scheduling with two kinds of VM-to-VM rules: affinity and anti-affinity rules. Affinity rules keep VMs together on the same host and anti-affinity rules separate VMs on different hosts. The vSphere PowerCLI is a command-line and scripting tool based on PowerShell that provides useful functionality for vSphere management. Throughout the rest of the section we discuss about managing a rule (i) to the DRS from the PowerCLI and (ii) to btrScript. This rule ensures that the VMs named *proxy1*, *proxy2* and *proxy3* do not run on the same host. Two other VMs with similar names (*proxy4* and *proxy5*) exist in the infrastructure that is annoying to use regular expressions used by both PowerCLI and btrScript tools.

The syntax for the rule with the PowerCLI tool is:

```
New-DrsRule -Name Proxy -Cluster cl1 -KeepTogether:$false
    -VM Proxy1,Proxy2,Proxy3
```

The first parameter is the name of the rule to identify it. As rules are associated to a cluster in the DRS, we assume all VMs run to the cluster *cl1*. The *KeepTogether* parameter defines if the VMs must run on the same host ($true) or if VMs must run on different hosts ($false). At the end, VMs are selected from their name.

The syntax for the rule with the btrScript language is:

```
Proxy: Proxy[1-3] spread
```

In the btrScript tool, rules are not associated to a cluster and so the cluster name does not appear in the rule declaration. The operator *spread* is used instead of the 'KeepTogether' parameter. In the btrScript language, each rule has one operator to keep clear confusion. So the *group* operator makes affinity rules and the *spread* operator makes anti-affinity rules. The syntax of the btrScript language allows regular expressions that is why the VM selection is shorter with btrScript.

The *New-DrsRule* command allows to enable or disable a rule at the declaration time. That is not allowed with btrScript. However, the DRS is an autonomic system where the administrator cannot solve rules when needed. In btrScript, the administrator explicitly calls the placement module enabling him to disable the rule before the rule resolution. Moreover the rule modification is easier with btrScript than with PowerCLI. The syntax to add the VM *Proxy4* to the previous rule in PowerCLI is:

```
Set-DrsRule -Rule Proxy -VM Proxy1,Proxy2,Proxy3,Proxy4
```

The administrator has to redefine the whole set of VMs. With btrScript, the administrator modify the VM set by including or excluding VMs:

```
Proxy:vms + Proxy4
```

If a rule is added or modified and it conflicts with another rules the vSphere policy is to disable the new one. As this management policy can hide issues, in btrScript, the insertion of a rule that conflicts with other ones is canceled and a notification showing conflicted rules is sent to the administrator. If he wants to insert the new rule, he has to disable conflicted rules.

Affinity rules between a group of VMs and a group of hosts also exist in vSphere. This VM-to-Host rules correspond to the *on* and *noton* rules in btrScript. Nevertheless their manipulation from the PowerCLI[6] is more complicated than the rules above described. Moreover there is no conflict detection for this kind of rules. So the administrator can insert one rule and its opposite without receiving any notification.

When an action is invoked by the vmWare administrator, placement rules are not verified. So actions can violate rules. In btrScript, administrators can not execute an action that violates one or more active rules. An error about the broken rules is reported.

To conclude brtScript and PowerCLI propose a similar approach of placement rules. However the rule management is easier from btrScript thanks to advanced selection mechanisms and more verifications, especially those on actions.

VII. Related Work

A. Business Rule Manager System (BRMS)

BRMS, like Drools [13], allow to set business rules to manage a system. However, these rules are simple with the syntax: "when something is true, do these operations". In btrScript, placement rules does not define actions to execute if the rule is not satisfied because rule satisfactions depend on the resource organization and other rules.

[5]http://www.vmware.com/support/developer/PowerCLI/

[6]http://www.van-lieshout.com/2011/06/drs-rules/

B. Virtual machine manager

VMWare [14] [15] can manage servers and VMs. Placement rules (called affinity rules in the VMWare documentation) are used to restrict placement between VMs and servers. These affinity rules include required and preferential rules. Required rules are similar to the btrScript rules and preferential rules can be violated to allow the proper functioning of the VMWare placement module. Preferential rules is excluded from btrScript but the definition of affinity rules from the VSphere GUI is not appropriate for managing large infrastructures. Select thousand VMs and servers in a GUI is not relevant. Administrators have to script themselves functions to add, remove and list affinity rules through the one of the vSphere API. Moreover, the consistency checking must be added for scheduled rules. In btrScript, these functions are integrated into the language operators. Further activation period for rules are not designed and VMWare actions does not take care of placement rules whereas btrScript does.

OpenNebula [16] is an open source toolkit for cloud computing designed to manage a large amount of VMs. A placement module called mm_sched (i.e., match making scheduler) allows to choose three different policies (compared with two policies implemented in btrScript) for the VM placement. However, these policies can not be tuned with specific rules.

C. Domain Specific Languages (DSL)

Puppet [17] [18] is a declarative configuration language for auditing and configuring large infrastructures (with virtualization or not) from one centralized node. A visual dashboard and reporting tools monitor servers to report every change. Puppet deploys large infrastructures but, at run-time, there is a lack of operations to handle servers or VMs.

VGrADS [19] and its virtual grid execution system allows to describe jobs with vgDL [20] and run them with time constraints. These tools address issues about deploying and scheduling jobs but, like Puppet, it is not designed to handle and perform reconfigurations on VMs and servers.

The former tools are designed to deploy and use resources of a virtual infrastructure while btrScript handles resources after deployment like Usher [21]. Usher is a shell for VM management. It is designed to local management and so it does not provide information about the whole grid.

Plasma [4] is the Entropy DSL to add constraints. It allows to define constraints by selecting VMs and servers from their name. However, physical and logical hierarchies do not exist and there is no selection on element properties. Moreover, no operation exists in the language to perform actions or query resource utilization.

VIII. Conclusion

Our paper presented btrScript, a safe management system for virtualized data center. It focuses on secure scheduled actions and placement rules. Scheduled actions allow to plan in advance tasks and schedule repetitive tasks for the automation of administrative tasks. Placement rules, which can be scheduled too, allow administrators to define the VM placement more accurately. These rules also restrict scheduled and immediate actions. A guardian module monitors the data center and reports issues such as overloaded servers and violated rules. We introduced a placement module named *Entropy* that enables to compute a plan with respect to rules and dynamic virtual and physical resources. This module can solve issues on all the data center or on a specific designated part. The evaluation is a comparison between the command line interface, PowerCLI, that can insert placement rules in the vmWare vSphere client. The btrScript system provides safer management by checking rules when an action is invoked and detecting more contradiction than the vSphere client.

Future work focus on running btrScript in a larger virtual infrastructure. BtrScript only runs with small architectures (20 servers) based on the kvm hypervisor. The use of the experimental platform grid5000[7] is planned to set up a large infrastructure.

IX. Acknowledgements

This work is partially funded by the SelfXL ANR/ARPEGE project (http://selfxl.gforge.inria.fr/dokuwiki/doku.php). We would like to thank Rémi Sharrock for their contributions.

References

[1] V. Soundararajan and J. M. Anderson, "The impact of management operations on the virtualized datacenter," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 326–337. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816003

[2] VMWare, "VMWare Infrastructure: Resource Management with VMWare DRS," VMWare, Tech. Rep., 2006.

[3] F. Hermenier, A. Lèbre, and J.-M. Menaud, "Cluster-wide context switch of virtualized jobs," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 658–666. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851574

[4] F. Hermenier, J. Lawall, J.-M. Menaud, and G. Muller, "Dynamic Consolidation of Highly Available Web Applications," INRIA, Research Report RR-7545, 02 2011. [Online]. Available: http://hal.inria.fr/inria-00567102/en/

[5] P. Horn, "Autonomic computing: IBM's Perspective on the State of Information Technology," 2001.

[7] https://www.grid5000.fr

[6] A. Computing, "An architectural blueprint for autonomic computing." *Quality*, vol. 36, no. June, p. 34, 2006. [Online]. Available: http://users.encs.concordia.ca/~ac/ac-resources/AC_Blueprint_White_Paper_4th.pdf

[7] R. Pottier, M. Léger, and J.-M. Menaud, "A reconfiguration language for virtualized grid infrastructures," in *10th IFIP international conference on Distributed Applications and Interoperable Systems (DAIS)*, vol. 6115, June 2010, pp. 42–55.

[8] R. Sirdey, J. Carlier, H. Kerivin, and D. Nace, "On a resource-constrained scheduling problem with application to distributed systems reconfiguration," *European Journal of Operational Research*, vol. 183, no. 2, pp. 546 – 563, 2007.

[9] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, July 2004.

[10] A. Goel and P. Indyk, "Stochastic load balancing and related problems," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*, 1999, pp. 579 –586.

[11] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, ser. MGC '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org.gate6.inist.fr/10.1145/1890799.1890803

[12] F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in virtualized datacenters," in *Proceedings of the 17th international conference on Principles and practice of constraint programming*, ser. CP'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 27–41. [Online]. Available: http://dl.acm.org/citation.cfm?id=2041160.2041167

[13] C. L. Forgy, "Expert systems," in *Expert systems*, P. G. Raeth, Ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, ch. Rete: a fast algorithm for the many pattern/many object pattern match problem, pp. 324–341. [Online]. Available: http://portal.acm.org/citation.cfm?id=115710.115736

[14] VMWare, "Resource management with vmware drs," VMWare, Tech. Rep., 2006.

[15] V. vSphere 4.1, "What's new in vmware vsphere 4.1 - availability and resource management," VMWare, Tech. Rep., 2010.

[16] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[17] L. Kanies, "Puppet: Next-generation configuration management," *j-LOGIN*, vol. 31, no. 1, pp. ??–??, Feb. 2006. [Online]. Available: http://www.usenix.org/publications/login/2006-02/pdfs/kanies.pdf

[18] D. Edwards*, A. Schafer†, T. Tyree†, A. Shortland*, A. Honor‡, and L. Thompson*, "Web ops 2.0: Achieving fully automated provisioning," DTO Solutions* and Puppet Labs† and ControlTier Project‡, Tech. Rep., 2009.

[19] L. Ramakrishnan, C. Koelbel, Y. S. Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T. M. Huang, K. Thyagaraja, and D. Zagorodnov, "Vgrads: enabling e-science workflows on grids and clouds with fault tolerance," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–12.

[20] Y.-s. Kee and C. Kesselman, "Grid resource abstraction, virtualization, and provisioning for time-targeted applications," in *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 324–331. [Online]. Available: http://portal.acm.org/citation.cfm?id=1371605.1372479

[21] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: an extensible framework for managing custers of virtual machines," in *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–15.