BtrPlace: A Flexible Consolidation Manager for Highly Available Applications

Fabien Hermenier, Julia Lawall, and Gilles Muller, *Senior Member, IEEE*

Abstract—The massive amount of resources found in datacenters makes it possible to provide high availability to multi-tier applications. Virtualizing these applications makes it possible to consolidate them on servers, reducing runtime costs. Nevertheless, replicated VMs have to be carefully placed within the datacenter to provide high availability and good performance. This requires resolving potentially conflicting application and datacenter requirements, while scaling up to the size of modern datacenters.
We present BtrPlace, a flexible consolidation manager that is customized through configuration scripts written by the application and datacenter administrators. BtrPlace relies on constraint programming and an extensible library of placement constraints. The present library of 14 constraints subsumes and extends the capabilities of existing commercial consolidation managers. Scalability is achieved by splitting the datacenter into partitions and computing placements in parallel. Overall, BtrPlace repairs a non-viable placement after a major load increase or a maintenance operation for a 5,000 server datacenter hosting 30,000 VMs and involving thousands of constraints in 3 minutes. Using partitions of 2,500 servers, placement computing is reduced to under 30 seconds.

Index Terms—Virtualization; Datacenter; Resource Management; Service Level Agreements; Reconfiguration

1 INTRODUCTION

MOST modern web applications, such as Facebook, Twitter, and eBay, are now structured as n-tier services, comprising, for example, a load balancer, an http server, a specific business engine, and a database. Such applications must be highly available (HA), which is achieved by replicating the tiers. Running these tiers in a datacenter makes spare hardware resources available for restoring an application to a viable state after a hardware failure. Using virtualization and dynamic consolidation allows tiers of multiple applications to be efficiently run on a single server; VMs with low activity are run together, and they are relocated to separate servers when their load increases [17], [21], [24].

Simply replicating tiers is, however, not sufficient to ensure high availability. The VMs running the replicas must also be placed on servers in such a way that there is no single point of failure. Several current consolidation managers provide this ability [5], [12]. However, in practice, there may be many other requirements that a VM consolidation manager must satisfy. For example, (i) replicas of stateful tiers may need to be placed on servers that are connected with low latency, to allow the use of consistency protocols without incurring excessive overhead, (ii) VMs running stateful tiers must be relocated using live migration [9] to maintain their current state, but other VMs may be relocated by booting a clone on another server, if doing so is more efficient, (iii) some VMs may require isolation. Furthermore, the datacenter administrator may need to express requirements related to the infrastructure management, such as leaving some servers free of VMs to allow maintenance. The set of requirements that need to be expressed may evolve over time with the emergence of new technologies and software architectures. For example, the widely used VMware vSphere and EC2 have already been updated to support additional constraints, such as VM-to-host affinity constraints in vSphere 4.1 [12] and dedicated servers in Amazon EC2 [1]. Before these releases, these requirements were not expressible. Finally, a consolidation manager must be highly scalable, as current datacentersuse servers housed in multiple shipping containers, holding up to 2,500 servers each.[1]

The above issues reveal that a consolidation manager to support HA applications must be flexible at two levels. First, the functionalities that the consolidation manager provides must be *configurable* by application and datacenter administrators, to be able to take into account the specific, and possibly changing, properties of the application and the datacenter, respectively. Second, the consolidation manager must be *extensible* to be able to support new functionalities as new requirements emerge. Most current approaches to designing consolidation managers use ad-hoc approximation algorithms [12], [21], [24]. These algorithms may be highly optimized to provide scalability, but they are not extensible and are difficult or impossible for application or datacenter administrators to configure. Some recent research solutions are based on combinatorial optimizations [5], [17], [18], [26], which are more configurable. Still, these approaches have not been explicitly designed

• *F. Hermenier is with the University of Nice Sophia-Antipolis, INRIA - CNRS - I3S. E-mail: fabien.hermenier@unice.fr*
• *G. Muller and J. Lawall are with INRIA/LIP6-Regal. E-mail: {gilles.muller,julia.lawall}@lip6.fr*

1. http://www.datacentermap.com/blog/datacenter-container-55.html

to be extensible, and they do not scale to infrastructures composed of thousands of servers.

Previous work [15] has discussed the theoretical foundations for a configurable and extensible consolidation manager, based on *constraint programming*. Here, we instantiate this design as a full-scale consolidation manager, BtrPlace (pronounced *BetterPlace*) which provides: (i) configurability, by providing a high-level scripting language allowing administrators to describe application and datacenter requirements. (ii) extensibility, by the use of Constraint Programming (CP), allowing new placement constraints to be easily developed and integrated; (iii) scalability, by exploiting any partitioning that is implied by the datacenter administrator constraints, allowing independent sub-problems to be solved in parallel. BtrPlace can also be used by the datacenter administrator to simulate various datacenter configurations and workloads, to help in dimensioning and maintenance tasks.

Our main results are:

Expressivity: the current library consists of 14 placement constraints that capture the needs of administrators in terms of reliability, isolation, performance and infrastructure management. These high-level placement constraints subsume and extend the placement strategies that can be expressed with the widely used consolidation managers VMware DRS and Amazon EC2.

Extensibility: The usage of CP makes placement constraints independent of each other. New constraints can be added without changing the existing implementation. The present constraints were implemented with an average of 30 lines of Java code each. Implementing new constraints required less than half a day of development.

Performance: On a simulated datacenter containing 5,000 servers hosting HA applications embedded into 30,000 VMs, BtrPlace can find a viable configuration in 3 minutes when it is facing a major load increase or maintenance. The number of application placement constraints has little impact on either the solving time or the quality of the reconfiguration plans.

Partitioning capabilities: BtrPlace analyzes the current configuration and the placement constraints to detect independent sub-problems that can be solved in parallel. Scalability is then limited by the number of servers available for computing solutions. Dividing a 5,000-server problem described above into two equal-sized partitions allows BtrPlace to find a viable configuration in 30 seconds.

The rest of this paper is organized as follows. Section 2 describes the design of BtrPlace. Section 3 describes its implementation. Section 4 evaluates our prototype. Finally, Section 5 describes related work, and Section 6 presents our conclusions and future work.

2 BtrPlace Overview

BtrPlace is a flexible consolidation manager that can be dynamically configured with placement constraints. Its task is to maintain the datacenter in a *viable* configuration, in which the placement of VMs on servers satisfies all of the constraints. BtrPlace can be customized through configuration scripts that describe application constraints for the placement of VM tiers and management constraints for the datacenter infrastructure. We first introduce the BtrPlace architecture and then present the configuration script language. Finally, we describe how BtrPlace performs VM re-assignment.

2.1 Architecture

BtrPlace runs on one or more VMs in the datacenter. It monitors the VM states and initiates the reassignment of VMs to servers when it detects that the current configuration is no longer viable. BtrPlace consists of four modules (see Fig. 1): monitoring, model merging, planning, and execution, that are executed within an infinite control loop.

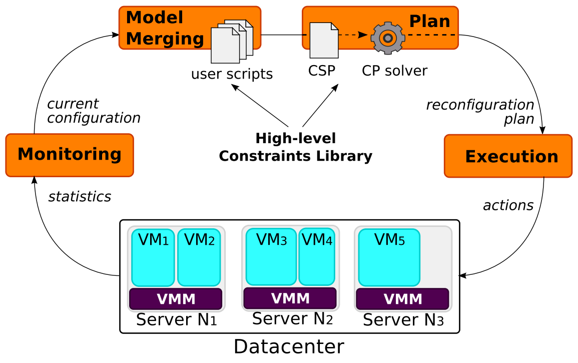


Figure 1: BtrPlace control loop.

Monitoring. The monitoring module retrieves the current state of each server and running VM, and each VM's current resource consumption and placement. The computing capacity of a server and the CPU consumption of a VM are expressed in terms of uCPU units, similar to the units used in Amazon EC2 [1], to abstract over heterogeneous hardware.

Model merging. The model merging module creates a configuration model by aggregating the current resource usage of all hosted applications and all the constraints on VM placement. The constraints on VMs come from the configuration scripts. These scripts are read at every loop iteration. A part of these constraints may be provided by provisioning strategies [22] developed by the application administrators to indicate the changing uCPU requirements for their VMs.The module then determines the viability of the current configuration according to the constraints in the configuration model. If all the placement constraints are satisfied, then the configuration is viable and BtrPlace restarts the control loop. Otherwise it continues to the planning module.

Planning. The planning module uses the configuration model to compute a new placement of the VMs and a *reconfiguration plan*, consisting of a series of actions that will relocate the VMs from their current servers to those indicated by the computed placement. Actions are scheduled to ensure their feasibility using an estimate of their duration. The planning module fails if some constraints cannot be satisfied. This situation reveals that the datacenter is no longer able to meet the application demand. To avoid this problem, BtrPlace can be used as a planning tool to limit the number of applications or to predict the need to acquire new servers.

Execution. The execution module applies a reconfiguration plan by performing all of the associated actions. Because BtrPlace creates the plan using timing estimates, some actions may not terminate within the anticipated time. If an action takes longer than expected, the execution module delays subsequent actions that depend on it. This can result in a longer reconfiguration, but cannot cause failure.

2.2 Configuration scripts

Configuration scripts permit the datacenter administrator and the application administrators to express their requirements on the VM placement and server state. A configuration script declares a set of servers or VMs and placement constraints to model a part of a viable configuration. The scripting language allows administrators to focus on defining a viable placement, rather than how to reach one, and to express their requirements without having to be concerned with the resource usage and expectations of others.

Describing a datacenter. A virtualized datacenter is composed of a collection of servers that run a Virtual Machine Monitor (VMM) such as Xen [3] to host VMs. Each VM runs either client applications or datacenter management services (monitoring system, resource manager, etc.). The datacenter administrator must describe the available servers and the connections between them. Servers are physically stacked into racks and interconnected through a hierarchical network [19]. Accordingly, there may be a non-uniform latency between them.

Listing 1 presents a BtrPlace script for describing a datacenter. Line 1 indicates the script name. The rest of the script describes properties of the servers, which are identified by their hostnames, prefixed with '@'. Variables are prefixed with '$'. Line 3 defines the variable `$servers` as the list of servers in the datacenter. Here, `$servers` is defined as a range of consecutive elements. Line 4 defines the variable `$racks` as the list of racks. Finally, line 6 makes `$racks` available to other scripts.

```
1 namespace datacenter;
2
3 $servers = @srv[1..12];
4 $racks={@srv[1..4],@srv[5..8],@srv[9..12]};
5
6 export $racks to *;
```

Listing 1: Description of a datacenter composed by 12 servers, grouped into 3 racks.

Describing an application. An application administrator must describe the application's VMs and their placement constraints. As in Amazon EC2, a *VM template* describes a basic VM disk image that is already configured for the datacenter. A template also specifies the amount of memory available to the VM and the VM's maximum CPU usage. To run a VM on a datacenter managed by BtrPlace, an application administrator must instantiate one of these VM templates. Once the VM instantiated, it can be customized by the application administrator to run the desired services.

Fig. 2 illustrates a typical 3-tier HA web application. The Apache services in tier T_1 and the Tomcat services in tier T_2 are stateless: all the handled requests are independent transactions and no synchronization of their state is required. On the other hand, tier T_3 runs a replicated MySQL database, which is stateful: transactions that modify the data must be propagated from one VM hosting a replica of T_3 to the others, to maintain a globally consistent state. In order to provide high availability, VM replicas of a tier must run on distinct servers. In order to ensure good performance, VM replicas of a stateful tier must be hosted on servers connected by a low latency, medium to reduce the synchronization overhead.

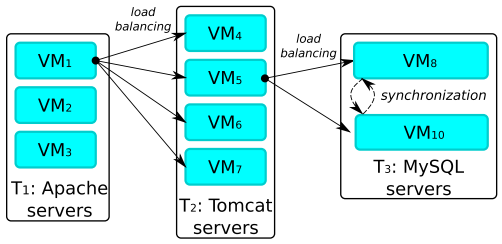


Figure 2: A 3-tier web application.

Listing 2 presents the BtrPlace script for describing this HA web application (`app1`). Line 2 imports the datacenter configuration script in order to access the exported variable `$racks`. Lines 4 and 5 ask for 9 VMs. Here, VM1 to VM7 are instances of the `small` VM template while VM8 and VM10 are instances of the `large` VM template. A template is parameterized by a set of options that indicate to BtrPlace the VM's characteristics. The `clone` option indicates that the VM can be moved by reinstantiating it on the destination server. Once the new cloned VM has booted, the old VM is turned off. This is possible for VMs hosting stateless tiers. In addition, all VMs, by default, can be moved using live migration [9]. The remaining options `boot` and `halt` indicate the time in seconds required to boot and halt the VM, respectively. These times depend on the software running in the VM, and can be provided by the application administrator to override the defaults found in the templates.

Lines 6 to 8 of Listing 2 define the variables `$T1`, `$T2`, and `$T3`, which describe the VMs associated with each tier, respectively. VMs in `$T1` are listed explicitly, VMs in `$T2` are defined in terms of a range, and VMs in `$T3` are defined using an enumeration. The script then defines constraints on the placement of the application's VMs. To ensure high availability, a `spread` constraint (line 11) specifies that the given VMs must be hosted on distinct servers at all times, including during reconfiguration. An `among` constraint (line 13) forces the given VMs to be hosted within one of the sets of servers given in the second argument. As a result, all of the VMs in `$T3` will be placed on servers in a single rack, to optimize their synchronization. Finally, the special variable `$me` is used to export all of the declared VMs (line 14).

```
1  namespace clients.app1;
2  import datacenter;
3
4  VM[1..7]: small<clone,boot=5,halt=5>;
5  VM[8,10]: large<clone,boot=60,halt=10>;
6  $T1 = {VM1, VM2, VM3};
7  $T2 = VM[4..7];
8  $T3 = VM[8,10];
9
10 for $t in $T[1..3] {
11     spread($t);
12 }
13 among($T3,$racks);
14 export $me to sysadmin;
```

Listing 2: Description of the HA application `app1` for the datacenter presented in Listing 1.

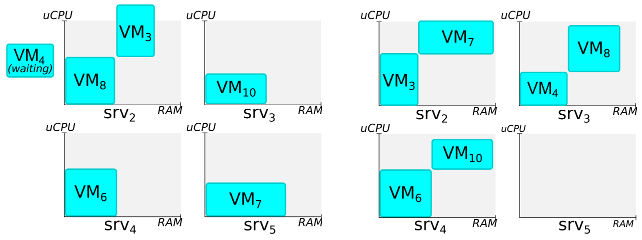
Administrating the datacenter. As illustrated in Listing 3, the datacenter administrator can also use configuration scripts to describe placement constraints related to the management of the infrastructure, such as datacenter partitioning, server maintenance, or the management of service VMs. In lines 2 and 3, the datacenter administrator imports the datacenter description and every script with a name starting with `clients`. For each imported script, a variable that lists the VMs declared in the script is automatically created. For example, the `$clients` variable contains every VM exported by the scripts starting with `clients`. In line 5, the administrator asks for one `large` VM that will be the service VM executing BtrPlace. In line 7, a `fence` constraint restricts `vmBtrPlace` to the server `srv1`. In line 8, a `lonely` constraint indicates that `vmBtrPlace` should not be collocated with any other VMs. Finally, in line 9, a `ban` constraint indicates that no client VMs should be hosted on `srv5`, to prepare it for a maintenance.

```
1 namespace sysadmin;
2 import datacenter;
3 import clients.*;
4
5 vmBtrPlace : large;
6
7 fence(vmBtrPlace,@srv1);
8 lonely(vmBtrPlace);
9 ban($clients,@srv5);
```

Listing 3: Datacenter administrator script

2.3 Reconfiguration Example

Given the datacenter described in Listing 1, the application described in Listing 2, and the datacenter administrator script in Listing 3, Fig. 3a depicts a non-viable VM configuration. In this case, (i) the uCPU demand of `VM3` is not satisfied because `srv2` does not provide sufficient uCPU resources for both `VM3` and `VM8`, (ii) `VM4` should be running, and (iii) `srv5` hosts `VM7` while its use has been banned by the administrator for maintenance.



(a) non-viable configuration (b) viable configuration

Figure 3: Partial sample configurations for the application described in Listing 2, running on the datacenter described in Listing 3. Each graph denotes the uCPU and memory capacity of a server. Each box denotes the estimated resource demand of a VM. Initially, VM$_4$ is waiting to be launched while the other VMs are running.

The planning module first computes a new viable configuration that satisfies the resource demands of the VMs and all of the placement constraints. To reach the new configuration shown in Fig. 3b, four actions must be performed: `VM4` must be launched on `srv3`, `VM10` must be moved to `srv4`, `VM8` must be moved to `srv3`, and `VM7` must be moved to `srv2`. To ensure the feasibility of the reconfiguration process, `VM7` cannot be moved to `srv2` until `VM8` is moved to `srv3` because `srv2` does not initially have enough free resources to accommodate it. In addition, moving `VM8` to `srv3` before moving `VM10` to `srv4` breaks the constraint `spread(VM[8,10])`.

The planning module also determines whether VMs should be moved by reinstantiation or by live migration. With an estimated migration duration of 20 seconds, and an estimated instantiation duration of 2 seconds, the planning module detects that it is more efficient to re-instantiate `VM7`, as the new replica will be available with sufficient resources in 7 seconds. However, for `VM8` and `VM10`, which have boot durations of 60 seconds, migration is preferable. A possible reconfiguration plan is shown in Table 1.

Start	End	Action
0'00	0'02	`instantiate VM7'`
0'00	0'05	`boot VM4 on srv3`
0'00	0'20	`migrate VM10 to srv4`
0'20	0'30	`migrate VM8 to srv3`
0'30	0'35	`boot VM7' on srv2`
0'35	0'42	`halt VM7 on srv5`

Table 1: Reconfiguration plan to reach the viable configuration in Fig. 3b from the non-viable one in Fig. 3a.

3 Implementing BtrPlace

The model merging and planning modules rely on a *core reconfiguration problem* (core RP) that models the current memory and uCPU demands of the VMs. This core RP is then extended with the *high-level placement* constraints provided by the datacenter and application configuration scripts, resulting in a *full reconfiguration problem* (full RP). BtrPlace relies on Constraint Programming (CP) [20] to model and solve RPs. The use of CP makes the core RP composable with administrator-provided placement constraints.

We first introduce CP and then describe the implementation of the core RP. We then present the library of high-level placement constraints. Finally, we address solver optimizations.

3.1 Constraint Programming

Constraint Programming is an approach to model and solve combinatorial problems in which a problem is modeled by stating constraints (logical relations) that must be satisfied by the solution. For a given problem and sufficient time, the CP solving algorithm is guaranteed to compute a globally optimal solution, if one exists. The algorithm is independent of the constraints composing the problem and the order in which they are provided. To use CP, a problem is modeled as a *Constraint Satisfaction Problem* (CSP), comprising a set of *variables*, a set of *domains* representing the set of possible values for each variable and a set of *constraints* that represent the required relations between the values of the variables. A solver computes a *solution* for a CSP by assigning each variable to a value that simultaneously satisfies the constraints. To make the solving process as scalable as possible, the challenges are to model the problem using the most appropriate basic constraints and to implement domain-specific heuristics to guide the solver efficiently to a solution.

BtrPlace uses the Java open-source constraint solver Choco.[2] The model merging module translates the configuration scripts into Choco constraints that are added to the core RP. Because Choco constraints are designed in terms of a number of standard constraints [4], BtrPlace will be able to benefit from improvements in the performance of CP, while it should also be possible to port the RP to other solvers.

3.2 Core Reconfiguration Problem

Computing a viable configuration requires first choosing for each VM a hosting server that satisfies the VMs resource requirements and then planning the actions that will convert the current configuration into the chosen one. This problem is expressed as the core RP. We now describe the variables, domains, and basic constraints that model the core RP. For reference, the variables are summarized in Table 2. We then propose a metric for estimating the cost of executing a reconfiguration.

Modeling the datacenter. The core RP is defined in terms of a set of servers $\mathcal{N}$ and a set of virtual machines $\mathcal{V}$. Each server $n_j \in \mathcal{N}$ is denoted by its memory usage n_j^{mem}, and its uCPU usage n_j^{cpu}, both capped by their respective maximum capacity. In addition, the boolean variable n_j^q, denotes the server state at the end of the reconfiguration process. It is instantiated to 1 if the server should be online at this point, and 0 otherwise.

Variables related to VM Management	
c^{host}	Current host of the VM (constant)
c^{men}, c^{cpu}	Current amount of memory and uCPU resources allocated to the VM (constant)
c^{ed}	Time the VM may leave its current host
d^{host}	Next host of the VM
d^{men}, d^{cpu}	Next amount of memory and uCPU resources to allocate to the VM
d^{st}	Time the VM arrives on its next host
Variables related to server management	
n^q	Next state of the server
Variables related to action management	
a^{st}, a^{ed}	Times an action starts and ends, respectively

Table 2: Variables composing the core RP.

We define a *slice* as a finite period during a reconfiguration process where resources are consumed on a server. A slice is represented as a collection of variables that are used within the core RP to represent the resource consumption of each VM and server throughout the entire duration of the reconfiguration process.

A consuming slice (*c-slice*) $c \in \mathcal{C}$ is a slice where resources are consumed at the beginning of the reconfiguration process. The variable c^{host} indicates the location of the slice. The variable is assigned to a value so that $c^{host} = j$ indicates the c-slice is hosted on the server n_j. Until the end of the slice, at the moment c^{ed}, a c-slice is considered to consume a constant amount of uCPU c^{cpu} and memory c^{mem}. A demanding slice (*d-slice*) $d \in \mathcal{D}$ is a slice where resources are consumed on some server at the end of the reconfiguration process. The variable $d^{host} = j$ indicates that the d-slice is hosted on the server n_j. The d-slice d starts at the moment d^{st} and ends at the end of the reconfiguration process, at the moment d^{ed}. During the whole duration of a d-slice, a constant amount of memory d^{mem} and uCPU d^{cpu} are then considered to be consumed.

Modeling the actions. The manipulation of the servers' states and the assignment of the VMs to servers may lead to the execution of actions. During a reconfiguration, a server can be turned off and on using the *shutdown* and the *boot* action respectively. A running VM may stay on the same server or may be moved to another server using a *migration* or a *re-instantiation* action. Finally, a waiting VM may be started on some server using a *start* action. The variables a^{st} and a^{ed} denote respectively the moments when an action $a \in \mathcal{A}$ starts and ends. The duration of an action can be estimated and modeled from experiments [16]. The datacenter administrator provides an estimate, parameterized by the properties of the involved VM or server, of the duration of each of the possible actions (migration, instantiation, or launch of a VM, start or shutdown of a server). As the duration of booting or halting a VM depends on the software installed by the application administrator, the

2. http://choco.emn.fr

estimated duration of these actions can also be provided in the configuration scripts.

The assignment of a waiting VM to a server is modeled using a d-slice and results in a launch action. Fig. 4 uses a Gantt diagram to illustrate a launch action a for `VM1` on server `N2`. When action a starts, the VMM allocates the memory for `VM1` and boots the guest OS. Once the guest OS is booted, the application starts and causes the VM to consume CPU resources. The d-slice continues to the end of the complete reconfiguration process, as it represents the fact that the VM is hosted by the server during this time. It may thus continue beyond the end of the estimated duration of the launch action.

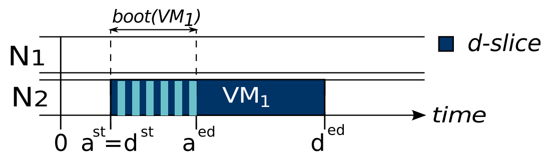


Figure 4: Modeling the launch of the waiting VM `VM1`.

The activity of a running VM is modeled using a c-slice on its host server and a d-slice on the server that will run it at the end of the reconfiguration process. If the d-slice and the c-slice are not placed on the same server, there will be a relocation. By default, the relocation is performed using a migration action. However, if the VM has the `clone` option, then BtrPlace uses the cost functions to determine whether a re-instantiation would be faster. When the relocation is performed, the c-slice and the d-slice overlap for a finite period, equaling the estimated duration of the action. Fig. 5 illustrates the relocation of `VM3` from `N2` to `N1` using live migration.

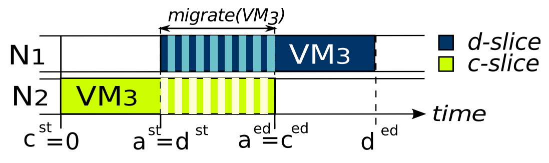


Figure 5: Modeling the migration of the running VM `VM3`. The action will start at the beginning of the d-slice and terminate at the end of the c-slice.

A server that can be booted is modeled using a c-slice that occupies all the server resources for a duration provided by the cost model. This ensures that no VMs are assigned to this server before booting has completed. To model the action of booting a server, we establish a relation between its state variable and its memory usage. We consider that a server is hosting VMs when its memory usage is not null. Its state variable is set to 1 if and only if its memory usage is greater than 0.

A server that can be shut down is modeled using a d-slice that does not use any resources. This allows the shutdown action to be scheduled with the other actions, but makes it possible to place other d-slices, and thus VMs, on the server if the server is allowed to stay online. However, if the server must be turned off, then no other d-slices should be placed on the server. This is modeled by a constraint that restricts the number of d-slices to 1 when the state variable of the server is 0. In this case, the starting time of the d-slice is set to the maximum of the finish times of the c-slices hosted on the server, and its duration is at least the estimated duration of the shutdown action provided by the cost function.

Satisfying the VMs' resource demand. To run the VMs optimally, in the final configuration, a server must not host slices with a total uCPU or memory demand greater than its capacity. The final configuration is determined by the placement of the d-slices. To place the d-slices with regard to their resource demand, we augment the core RP with *bin-packing* constraints. A *bin-packing* constraint packs items with a specific size into bins with a finite capacity. As a bin-packing constraint only accounts for one resource, we use two instances, for the uCPU and for the memory.

The satisfaction of VMs bandwidth requirements is not addressed in this paper. To support high-speed network requirement at large scale, datacenters have started to rely on network fabrics having a full bisection bandwidth [13], [14]. In this setting, a bandwidth constraint violation only occurs when a server hosts VMs that have their cumulated bandwidth requirements exceeding its capacity. Satisfying bandwidth requirements can then be modeled similarly to the support of memory and uCPU requirements.

Scheduling the actions. A server may be the source of outgoing relocations and the destination of incoming relocations and launch actions. To ensure the feasibility of reconfiguration, incoming actions can only be executed when there are sufficient free uCPU and memory resources on the server. Thus, some outgoing actions may need to be executed prior to some incoming actions, to free the required resources. To plan the execution of the actions, the standard constraint *cumulatives* may be used to choose the actions' start and finish times to schedule the incoming actions according to the outgoing actions. As this constraint is not available in the Choco library, we have implemented a dedicated version.

Dependencies between relocations may be cyclic. This may occur when the placement of the VMs is computed without considering how the migrations can be scheduled. A solution is to add a relocation to a temporary pivot server to break the cycle [2], [17] after the placement has been computed. As this extra relocation slows down the reconfiguration process, cyclic dependencies are generally undesirable. To avoid cycles, our model allows at most one relocation per VM and both selects the VM placement and schedules the actions in a single core RP. If the solver tries to place a VM on a server that will lead to a cycle, the scheduling constraint will invalidate this placement, causing the solver to directly place the VM on what would be the pivot server.

Solving a RP then requires computing a value for each d^{host} and a^{st}. The chosen values indicate how to create a reconfiguration plan containing the actions to perform and a time for each action to start that ensures the action's feasibility with regard to its estimated duration.

A given RP may have multiple solutions, varying in terms of the costs entailed. Performing a reconfiguration takes non negligible time, during which the performance of running applications may be impacted until the reconfiguration is completed. In addition, the duration of a migration and a re-instantiation increases with the memory usage of the migrated VM and its booting duration, respectively. Finally, a relocation consumes resources on the involved servers and thus temporarily decreases the performance of applications, if the servers do not have spare capacity. A good reconfiguration plan is thus composed of a few, fast actions, where most of them are executed in parallel, with a minimum delay [17], [24]. Given a notion of cost, a constraint solver can compare possible solutions, and return the one with the lowest cost. The cost of a reconfiguration is defined as the sum of the elapsed time between the moment when the reconfiguration starts and the moment when each reconfiguration action has completed. This measure accounts for the number of actions, their execution time, and their delay. It is expressed in Choco by defining the *objective* variable $K = min(\sum_{a \in \mathcal{A}} a^{ed})$. To compute K, Choco works incrementally: each time a solution x with an associated cost k_x is computed, Choco automatically adds the constraint $K < k_x$ and tries to compute a new solution. This added constraint ensures the next solution y will have a cost $k_y < k_x$. This process is repeated until Choco browses the whole search space or hits a given timeout. It then returns the last computed solution.

3.3 Script Placement Constraints

Administrators use configuration scripts to express placement requirements. BtrPlace provides an extensible library of placement constraints designed according to a modular architecture. Each constraint is implemented as a module that provides the signature of the placement constraint and an implementation that interacts with variables from the core RP to restrict the VMs' resource allocation, their placement, server states, or the action schedule. The current library provides 14 constraints targeting either the datacenter administrator or application administrators and focusing on performance, reliability and partitioning concerns (see Table 3).

`Gather` forces all of the given VMs onto the same server. This constraint is relevant for application administrators who want to minimize the network communication between their VMs. The constraint is implemented by forcing all the placement variables of the d-slices to take the same value.using *eq* constraints.

`Spread` forces the given VMs onto distinct servers, even during reconfiguration. To force the VMs to be hosted on distinct servers at the end of the reconfiguration process, `spread` forces the d-slice placement variables to each take on distinct values. To prevent the specified VMs from overlapping on any server during the reconfiguration process, the arrival time of a d-slice is increased to at least the departure time of a c-slice if their placement variables are equal.

`Among` ensures that the given VMs are hosted on a single set of servers among those given. The implementation forces that if any d-slice of one of the given VM is hosted on a server, then all the given VMs are hosted on the set of servers to which this server belongs.

`Ban`, `fence`, `quarantine`, and `root` restrict the set of servers considered for a given set of VMs. `Ban` prevents the use of the given servers by the VMs to help the datacenter administrator preparing software main tenant on the servers. `Fence` limits the VMs to the given servers. It enables the datacenter administrator to partition the datacenter. `Quarantine` creates a zone where no VMs can enter or leave to limit the contagion when some servers appear to be compromised. `Root` prevents the VMs from being relocated, to make BtrPlace compatible with infrastructure that does not support relocation. These constraints rely on domain restrictions: for each of the given VMs, the disallowed servers are removed from the domain of each d-slice placement variable.

`Lonely` and `splitAmong` address isolation issues between VMs. `Lonely` keeps the given set of VMs separate from all other VMs, allowing an application administrator to prevent critical VMs from being hosted with potentially suspicious VMs. Its implementation forces the set of d-slice variables of the given VMs and the set of d-slice variables of all the other VMs not to have any values in common. `SplitAmong` forces several sets of VMs to be hosted on distinct set of servers. This constraint enables for example, disaster recovery by forcing tier replicas to be hosted on distinct racks so that even in the case of a major outage affecting a whole rack, the application will still be available. The implementation of `splitAmong` relies on some of the principles of the `among` and the `spread` constraints.

`Capacity` restricts the total number of VMs hosted on a set of servers. It enables the datacenter administrator to control the use of shared resources, such as public IP addresses. Its implementation limits the number of d-slice placement variables that are assigned to the servers.

`Preserve` and `oversubscription` control the resource demand of the VMs and the servers' resource allocation, respectively. They allow the datacenter administrator and the application administrators to ensure that small changes in resource usage will not trigger a reconfiguration. Such an overallocation of resources to prevent frequent reconfigurations is standard industry practice. `Preserve` provides a minimum amount of uCPU resources to each of the given VMs. It enables an application administrator to prevent temporary resource outages due to load spikes. This constraint is implemented by increasing the uCPU resource required by the d-slice of each of the given VMs, if it is below the given threshold. `Oversubscription` ensures for each of the VMs hosted on the given servers an amount of uCPU resources equal to the VM's maximum usage

Placement Constraint	User		Primary concern			Manipulated elements		
	Hoster	Hostee	Resource	Reliability	Partitioning	Server	VM	Actions
`spread(vs)`		x		x			x	x
`gather(vs)`		x	x				x	
`among(vs,{`$\mathtt{ns}_1$`, ...,`$\mathtt{ns}_x$`})`		x			x		x	
`splitAmong({`$\mathtt{vs}_1$`,...,`$\mathtt{vs}_i$`},{`$\mathtt{ns}_1$`,...,`$\mathtt{ns}_x$`})`		x		x	x		x	
`ban(vs, ns)`	x				x		x	
`fence(vs, ns)`	x				x		x	
`root(vs)`		x		x			x	
`lonely(vs)`		x			x		x	
`quarantine(ns)`	x				x		x	
`capacity(ns, n)`	x		x				x	
`preserve(vs, n)`		x	x				x	
`oversubscription(ns, n)`	x		x				x	
`offline(ns)`	x		x			x		
`noIdles(ns)`	x		x			x		

Table 3: Placement constraints available in BtrPlace. Hoster and Hostee denote the datacenter administrator and an application administrator, respectively. `ns`, `vs` denote a set of servers and VMs, respectively. `n` denotes an integer.

reduced by the given ratio. For example, with a ratio of 0.8, each server must be able to provide each of the VMs it hosts with 80% of the VM's maximum uCPU usage, even if the VM's current demand is inferior. This constraint avoids placing too many VMs that currently consume a few uCPU on a single server, which could increase the chances of having a saturated server if the VMs simultaneously increase their uCPU demand. `Oversubscription` is implemented using a *bin-packing* constraint similar to the one in the core RP.

`Offline` and `noIdles` control the servers' states. `Offline` forces a set of servers to be offline for example, to allow the datacenter administrator to prepare for hardware maintenance. `NoIdles` causes the given set of servers to be turned off if they do not host any VMs. This constraint allows the datacenter administrator to provide a power saving policy. These constraints are implemented by manipulating the boolean state variable of each server in the set given as argument.

3.4 Optimizing the Solving Process

Computing a solution for the full RP may be time consuming for large datacenters, as selecting a host for each VM and planning the actions is NP-hard. BtrPlace uses three strategies to optimize the solving process: simplification of the full RP, heuristics guiding the CP solver toward fast reconfiguration plans, and partitioning.

Our first strategy, referred to as the *filter* optimization, is to limit the set of VMs that are considered by the generated full RP. Initially, each placement constraint uses a dedicated algorithm to check the viability of the current configuration and on failure computes a set of *candidate* VMs to try to replace, that is expected to be sufficient to compute a solution. For example, the `spread` constraint checks if the VMs to spread are already on distinct servers. If not, it selects as candidate VMs those which share servers. Only the VMs in the *candidate* set are considered to be relocatable.

Our second strategy is a heuristic that indicates to the solver the variables it has to instantiate first and the values to try for these variables. As moving larger VMs first reduces resource fragmentation, the solver considers the VMs in descending order of their memory usage. If the *filter* optimization is enabled, the solver focuses first on the VMs whose current placement violates at least one constraint. Each time a VM is selected, the solver first tries to place the VM on its current host to avoid useless relocations. In case of failure, if the *filter* optimization is enabled, the solver tries a server that does not host any of the candidate VMs selected by the *filter* optimization. As no resource will be freed on these servers during the reconfiguration process, relocation to such a server is guaranteed not to be delayed. Finally the solver tries the remaining servers. Servers having the largest amount of free memory are preferred in order to balance the VM load in the datacenter and to reduce the chance of having overloaded servers in the future.

Our third strategy is to identify independent sub full RPs to solve in parallel. For example, the datacenter administrator can split the infrastructure into several physical partitions, aligned with the network topology, and use `fence` and `among` constraints to restrict the placement of each application to one physical partition. Before the full RP generation, each constraint checks if the involved VMs and servers can be isolated from others to form a partition, or can join an existing partition. Typically, constraints such as `fence`, or `among` that restrict the placement of VMs to a known set of servers lead to the creation of new partitions. Constraints restricting the placement of VMs with regard to other VMs do not participate in partition creation. They are however analysed to check that they are consistent with the computed partitions. For example, if some of the VMs in a `spread` constraint are in different partitions, then the constraint is split so that it does not overlap multiple partitions.

BtrPlace generates one full RP for each of the partitions and solves them in parallel, or in a distributed manner. This decomposition may, however, fragment resources and cause some sub-problems to have no solution on a heavily loaded infrastructure.

4 Evaluation of BtrPlace

The goal of BtrPlace is to allow the specification of a large range of placement constraints and to repair non-viable configurations with regards to the specified constraints in the presence of disruptions, such as maintenance or long-lasting load increases. We first discuss the expressivity of BtrPlace for supporting various placement constraints. We then demonstrate the practical benefits of BtrPlace for easing maintenance on a small cluster. Finally, we evaluate the impact of placement constraints on the solving process of BtrPlace for a real-sized modern datacenter with up to 5,000 servers running 30,000 VMs.

4.1 Expressivity and Extensibility of BtrPlace

The current library of BtrPlace provides 14 placement constraints related to resource management, isolation, fault tolerance, and server management. These constraints cover all the affinity constraints in VMware DRS while `splitAmong` and `lonely` implement the two placement constraints available to EC2 clients: availability zones and dedicated instances, respectively. Table 4 compares the placement constraints in BtrPlace with those in VMware DRS [12]. `Fence` and `ban` are similar to the DRS VM-host affinity constraints while `gather` and `spread` are similar to the DRS VM-VM affinity constraints. However, `spread` additionally disallows the given VMs to collocate even during the reconfiguration. The `capacity` constraint can be emulated in DRS but doing so requires modifying environment variables. Capacity can furthermore only be limited for all servers in an entire cluster, rather than for an arbitrary set of servers, as in BtrPlace. To the best of our knowledge, `among`, `quarantine`, `oversubscription`, and `splitAmong` are not available in either DRS or EC2.

Constraint	Status	Constraint	Status
ban	=	fence	=
gather	=	spread	>
root	=	preserve	=
offline	=	noIdles	=
capacity	>	quarantine	x
among	x	splitAmong	x
lonely	x	overSubscription	x

Table 4: Compatibility of the constraint with regards to VMware DRS. 'x', '>', and '=' indicate that the BtrPlace constraint is not available, is superseded, or is equivalent to a constraint in VMware DRS, respectively.

The translation of the placement constraints into Java is straightforward and concise. It primarily consists of directly encoding the formal definition of the constraint into the Choco library and the BtrPlace API. On average, 30 lines of code (Loc.) are required to implement the constraints provided with BtrPlace (Table 5). These implementations have furthermore been designed to be as efficient as possible. Each placement constraint carefully analyzes the state of the variables in the core-RP to generate few and efficient Choco constraints.

Constraint	Loc.	Constraint	Loc.	Constraint	Loc.
spread	50	root	11	preserve	10
among	40	lonely	17	overSubscription	40
ban	20	quarantine	40	offline	10
fence	58	capacity	64	noIdles	10
gather	11	splitAmong	31		

Table 5: Lines of Java code per placement constraint.

The high-level expressivity and conciseness of Choco and the BtrPlace API allow the implementation of relevant placement constraints in only a few hours for an experienced developer. For example, `lonely` and `splitAmong` were not present in the initial version of BtrPlace, but were motivated by an extension to EC2 [1] and by a request from a BtrPlace user, respectively. Browsing the Choco API to choose the right basic constraints, and performing the implementation and the unit tests required half a day for each of these placement constraints. Neither the existing placement constraints nor the core RP were modified.

BtrPlace is currently used by the European project Fit4Green.[3] One goal of this project is to place VMs in a federation of clouds, taking into account energy concerns. Fit4Green developers having no background in Constraint Programming implemented a non-invasive extension of BtrPlace to provide constraints that restrict the hosting capacity of the servers, satisfy the hardware requirements of specific VMs, and reduce the global energy consumption of the clouds. [11]

4.2 Using BtrPlace to Prepare for Maintenance

We first study reconfigurations that restore a cluster to a viable configuration when an administrator needs to perform maintenance on a server. For this, we test three instances of the RUBiS benchmark [8], an auction site prototype that is modeled after *eBay.com*. Each instance of RUBiS provides a 3-tier web application and a benchmark to evaluate its performance. The evaluation platform is a small cluster, composed of 8 working servers. (`WN1` to `WN8`) and 4 service servers connected through a Gigabit network. All the servers have a 2.1 GHz Intel Core 2 Duo and 4 GB RAM. Each working server runs Xen 3.4.2 with a Linux-2.6.32 kernel and provides a capacity of 2.1 uCPU and 3.5 GB RAM for the VMs. Monitoring is performed by Ganglia.[4] Three of the service servers export the RUBiS VM images and run the benchmarks. The fourth service server runs BtrPlace.

We have tailored the RUBiS configuration so as to deliver the maximum possible performance. The three tiers of each instance of RUBiS are deployed as 7 VMs (for a total of 21 VMs), as described in Listing 4. Each VM in tier `$T1` has 512 MB RAM and runs Apache 2.1.12. Each VM in tier `$T2` has 1 GB RAM and runs Tomcat 7.0.2. Finally, each VM in tier `$T3` has 1 GB RAM and runs MySQL-cluster 7.2.5. This tier is stateful, and

3. http://www.fit4green.eu
4. http://www.ganglia.info

the coordination between the databases is provided by the MySQL's *ndb* engine. Requests involving dynamic content are distributed by each Apache service to Tomcat services using mod_jk 1.2.28. SQL queries executed in a Tomcat service are distributed to MySQL services using Connector/J 5.1.13. The initial deployment of the VMs is compatible with their placement constraints. Our experiment adds `ban` constraints at various times to allow for the maintenance of a server, and removes them later when the server is up again. At the same time, the server load may change due to the running benchmarks. Table 6 shows, for each of these events, the number of relocations performed and their duration.

```
//Datacenter description
...
$racks = {@srv[1..4], @srv[5..8]};

//Application description
$T1 = VM[1..2];
$T2 = VM[3..5];
$T3 = VM[6..7];
for $t in $T[1..3] {
    spread($t);
}
among($T3, $racks);
```

Listing 4: BtrPlace script for the environment.

Time	Event	Reconfiguration
2'10	+ `ban({WN8})`	3 + 3 relocations in 0'42
4'30	+ `ban({WN4})`	2 + 7 relocations in 1'02
7'05	− `ban({WN4})`	no reconfiguration
11'23	+ `ban({WN4})`	*no solution*
11'43	− `ban({WN8})` + `ban({WN4})`	2 relocations in 0'28

Table 6: Experimenting with maintenance. '+' indicates a constraint injection while '−' indicates a removal.

The `ban` constraints added in our experiment most obviously require relocating the VMs hosted on the specified servers to other servers. Nevertheless, such a constraint can also induce other relocations, when the amount of resources currently free on other servers is not sufficient for these VMs. This is illustrated at time 2'10, when the constraint `ban({WN8})` is injected. The addition of this constraint requires relocating the 3 VMs hosted on `WN8`, but it also requires three induced relocations to obtain a viable host for all of the VMs. A similar situation occurs at 4'30 when 7 induced relocations were needed to be able to migrate the 2 VMs that were running on `WN4`. Such induced relocations are hard to plan manually.

The end of the experiment illustrates the case when the full RP is unsolvable. At time 11'23, the constraint `ban({WN4})` is injected by the administrator. However, BtrPlace indicates that it is impossible to have a viable configuration in this situation. Logs revealed a load increase occurred after time 7'05. This increase was absorbed through migrations of VMs to WN4, making the `ban` insertion impossible to satisfy. At time 11'43, the administrator removes the ban on `WN8` and tries the ban on `WN4`, which is now successful, leading to 2 relocations.

4.3 Scalability of BtrPlace

The scalability of BtrPlace is determined by the time to reconfigure the datacenter, which comprises the time to solve the full RP and the time to apply the resulting reconfiguration plan. As compared to the core RP, the full RP includes the placement constraints, which reduce the possible solution space and thus may increase the solving time. These additional constraints may furthermore introduce the need to sequentialize some induced relocations, which may increase the reconfiguration time. To analyze scalability we generate non-viable configurations that simulate server failures or load increases and assess the resulting reconfiguration process.

We are not aware of established consolidation workloads that allow comparison with existing work. Furthermore, commercial consolidation manager implementations are proprietary [1], [12], while reimplementing related approaches from high-level descriptions in research papers [5], [18], [26] is unlikely to give realistic results. Accordingly, we rely on a workload generated from experimental data and industry best practice.

We simulate a datacenter with 5,000 servers that are similar to those in the *griffon* cluster, a part of the Grid'5000 testbed [7] that can be used to run VMs. Each server has a dual processor 2.5 GHz quad-core Intel Xeon-L5420 and 16 GB RAM. On the simulated datacenter, these servers are grouped by 250 on edge network switches and they provide 20 uCPU each.

In a datacenter, each hosted application has its own structure. Accordingly, we randomly generate the structure of each virtualized application. An application has 3 tiers and uses between 6 and 30 VMs, with at least 2 VMs per tier. 12 templates are available to instantiate VMs that require between 1 and 3 GB RAM and at most 30 to 60 uCPU. All the VMs in the same tier instantiate the same template. Each application can be run with or without placement constraints. When present, the placement constraints are expressed as one `spread` constraint per tier and one `among` constraint for the third tier. The uCPU consumption of each application is chosen randomly between 20% and 90%.

To illustrate the impact of the resource usage and the number of VMs and applications on BtrPlace, we vary the consolidation ratio on the servers from 3 VMs to 6 VMs amounting to up to 1,700 applications running a total of 30,000 VMs. This makes the overall memory and uCPU usage vary from 36% to 73% while 6 VMs per server is a common ratio observed in industrial datacenters hosting services [23]. For each consolidation ratio, we generated 50 different RPs that only differ by the initial VM placement and their resource allocation.

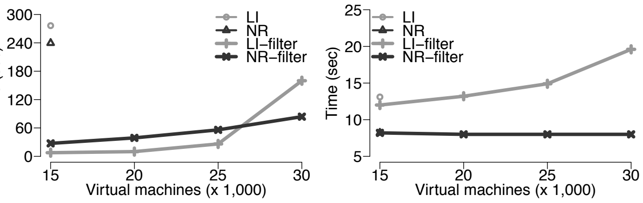
Finally, the action durations were estimated from measurements inside the *griffon* cluster. Table 7 summarizes the time to boot or halt a VM depending on the running service. Instantiating a VM from its template requires 2 seconds while a live VM migration requires 1 second per 100 MB RAM [17].

Embedded Service	Action duration (in sec.)	
	`boot`	`halt`
Apache	4	5
Tomcat	9	10
MySQL	5	6

Table 7: Estimated action duration.

We consider two scenarios: `LI` that simulates Load Increases and `NR` that simulates a maintenance for Network Rewiring. In `LI`, 10% of the applications are selected to have their uCPU demand increased by 30% (capped at 100%). This increases the overall demand by an average of 5% which induces the relocation of some VMs to restore a viable configuration. In `NR`, 5% of the servers are randomly selected to be shutdown for maintenance. This percentage is derived from observations reported on Google's datacenters where administrators have to rewire at any given moment, 5% of their servers. [10] In this scenario, The VMs hosted on these servers then have to be restarted elsewhere. We run BtrPlace on one core of a Intel Xeon X3440 at 2.53 GHz with 16 GB RAM that runs Linux 2.6.32-5-amd64 and Sun's JVM 1.6.0u26. For each experiment, we give the planning module 5 minutes to solve a full RP. The planning module succeeds at solving a RP when it computes a reconfiguration plan or proves that no solution exists within the allotted time.

Impact of the *filter* optimization. The *filter* optimization, defined in Section 3.4, reduces the number of VMs considered in a RP. In this experiment, we evaluate its practical interest on the core-RP, and thus no applications have placement constraints. Fig. 6 shows the impact of the *filter* optimization on the solving process.



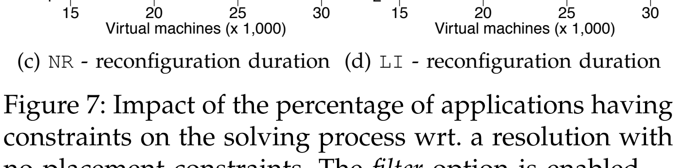
(a) Solving duration (b) Reconfiguration duration

Figure 6: Impact of the *filter* optimization

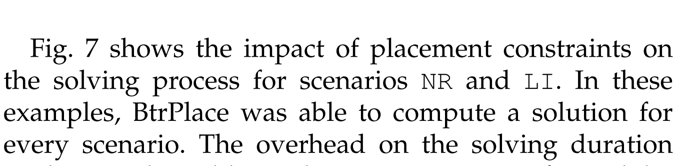
Fig. 6a shows that the solving duration for the RPs is exponential with respect to the number of VMs managed by BtrPlace. Indeed, solving a RP is NP-Hard. Without *filter*, BtrPlace was not able to compute a placement for 20% of the RPs having 15,000 VMs in the `LI` case. The solver has to select a host for each of the VMs which leads to big RPs that cannot be solved due to running out of memory or exceeding the alotted time. Enabling *filter* makes BtrPlace generate smaller RPs that were all computable by the solver. In `NR`, *filter* restricted the full RPs to the VMs that have to be restarted after failure. In `LI`, *filter* restricted the full RPs to the VMs hosted on the overloaded servers (3,000 VMs on average for RPs having 30,000 VMs). By reducing the number of considered VMs, *filter* can significantly reduce the average solving time and manage bigger RPs. In `LI`, with RPs having 15,000 VMs, the time is reduced from 276 seconds to 8 seconds. In `NR`, it is reduced from 240 seconds to 27 seconds. In the `LI` case, we also observe that the reconfiguration duration increases with the number of VMs. The overall load of the datacenter increases and it becomes more complicated for the solver to find a place for a VM that is immediately available. The solver determines that more VMs have to be relocated in advance to liberate enough resources to be able to reach a viable configuration.

Enabling *filter* leads to slightly faster reconfiguration plans (Fig. 6b). The option helps the solver in placing VMs on servers that can host the VMs without any delay. *Filter* thus reduces the number of induced relocations, and thus the number of actions and the estimated reconfiguration duration. Without *filter* option, 7.5% of the computed plans for RPs having 15,000 VMs contain delayed actions with a maximum reconfiguration duration equal to 32 seconds. Enabling the option enables the solver to compute plans without delayed actions and with a maximum reconfiguration duration equal to 14 seconds. In the `NR` case, *filter* does not help BtrPlace compute better reconfiguration plans. In this case, the reconfiguration is motivated by the fact that some servers have failed. BtrPlace is obliged to reboot the VMs that were on these servers, and only has the latitude choose the VMs affected by the induced relocations, if any. However, for the `NR` scenarios, the computed reconfiguration plans have no induced relocations even without *filter* and stay constant for all the RPs.

Impact of the placement constraints. In this experiment, we vary the percentage of applications having placement constraints. The *filter* optimization is enabled.



(a) `NR` - solving duration (b) `LI` - solving duration



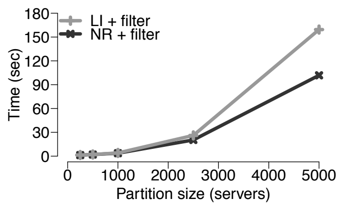
(c) `NR` - reconfiguration duration (d) `LI` - reconfiguration duration

Figure 7: Impact of the percentage of applications having constraints on the solving process wrt. a resolution with no placement constraints. The *filter* option is enabled.

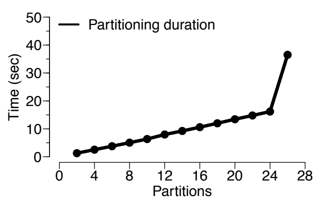
Fig. 7 shows the impact of placement constraints on the solving process for scenarios `NR` and `LI`. In these examples, BtrPlace was able to compute a solution for every scenario. The overhead on the solving duration is due to the additional computations performed by an increasing number of placement constraint. In the worst case, when all the applications have placement constraints, the overhead is 21% (18 sec) for RPs having 30,000 VMs in the `NR` case (Fig. 7a). However, for every RP, the dominating factor in the solving process remains the two *bin-packing* constraints in the core RP. In the `LI` case, the overhead is negligible or negative (Fig. 7b). As compared to the `NR` case, there are fewer VMs to place. When more than 66% of the applications have constraints, RPs with 30,000 VMs are faster to solve than RPs with 25,000 VMs. This reveals a *phase transition*. With this amount of constraints, the full RP is more complex to solve, but the filtering provided by each constraint helps the solver compute a solution more quickly. Finally, the placement constraints do not impact the estimated reconfiguration duration in the `NR` case. In the `LI` case, the reconfiguration duration increases when 66% of the applications have constraints but the phase transition limits this increase once this threshold is exceeded.

Impact of partitioning. We evaluate the impact of partitioning in the context of a datacenter having 5,000 servers and 30,000 VMs, `fence` constraints are injected into the full RP to force the creation of equal-sized partitions, varying from 250 to 5,000 servers. The *filter* option is enabled and all of the applications have placement constraints. A solution was found for every RP, except for two in the `LI` case and one in the `NR` case, the three having partitions of 250 servers. As all of the VMs of a given application must fit within a single partition, the use of small partitions may introduce resource fragmentation and reduce the datacenter hosting capacity.

Fig. 8a shows that partitioning the datacenter into smaller full RPs and solving them in parallel gives a significant benefit. For instance, solving a single partition of 5,000 servers requires 159 seconds for `LI`, while splitting the servers of the datacenter into two partitions of 2,500 servers each reduces the solving duration to 26 seconds. The partitioning process did not alter the computed reconfiguration plans and the estimated reconfiguration duration stays similar to the values reported in Fig. 7 when 100% of applications have constraints.



(a) Solving duration with 5,000-servers



(b) Partitioning duration with 2,500-server partitions

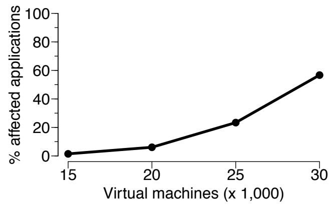
Figure 8: Partitioning impact.

Detecting partitions induces an overhead in the planning module. A current trend in datacenter design is to use multiple shipping containers, each storing up to 2,500 interconnected servers. To evaluate the partition detection overhead in the context of a datacenter having several containers, each corresponding to a partition, we generate configurations made up of a varying number of 2,500-server partitions. Fig. 8b shows that the duration of the partitioning process increases with the number of partitions. It is indeed correlated with the number of servers, VMs and constraints and as is necessary to check and check that none of the constraints overlap multiple partitions. Below 24 partitions, the partitioning duration is still lower than solving a single 2,500-server full RP for the worst observed case, `LI`. 24 2,500-server partitions results in a datacenter made up of 60,000 servers and 360,000 VMs. For more scalability, the partitioning process could itself be split among multiple servers. The scalability of BtrPlace is then limited by the number of sub-RPs it can simultaneously solve.

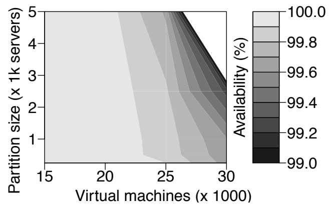
Availability provided by BtrPlace. We finally analyze the global availability rate provided by BtrPlace in the `LI` case. In this case, 10% of the applications ask for more resources. This saturates some servers and creates a non-viable configuration. Every application with at least one VM running on a saturated server then has its quality of service affected by the load increase.

The *Mean Time To Repair* (MTTR) denotes the average time to repair all the constraints of an application. An un-affected application as a repair time equals to 0. Otherwise, it equals R, the duration to solve the RP plus the reconfiguration duration. The MTTR is then expressed as $\alpha \times R$ where α denotes the ratio of affected applications. Fig. 9a shows the evolution of α depending on the number of running VMs.

Amazon EC2 relies on a hourly billing period. An application administrator should then provision the application's VMs correctly for an hour to minimize the hosting costs. If we consider that a load increase occurs after a period P of one hour, then we can compute the global availability provided by BtrPlace as follows: $\frac{P}{MTTR+P}$. Fig. 9b denotes the global availability provided by BtrPlace depending on the number of running VMs and the partition size.



(a) Affected applications



(b) Global availability

Figure 9: Availability in the `LI` case.

Reducing the partition size or the number of running VMs reduces the importance of the solving time in the MTTR. Reducing the number of running VMs also reduces the percentage of affected applications. To reach a specific availability rate, the datacenter administrator has to establish a trade-off between a high consolidation ratio, to host numerous VMs, and large partitions, to reduce resource fragmentation. These values can be estimated offline with BtrPlace and a workload representative of the datacenter usage. In the `LI` case, when all the applications have constraints, we observe that no partitioning is needed to reach an availability rate of 99.9% when there are fewer than 21,000 VMs. To host up to 25,000 VMs with an availability rate of at least 99.8%, partitions of 2,500 servers are needed.

In studying the results of our experiments, we observed that every plan contains at least the relocation of a VM running a Tomcat service which takes 9 seconds on our platform. This is thus a lower bound for the MTTR, and is not under the control of BtrPlace. With 25,000 VMs and partitions of 2,500 servers, this theoretical minimal reconfiguration duration represents 25% of the MTTR. Having faster actions, due to a better infrastructure for example, would then increase significantly the availability provided by BtrPlace.

5 Related Work

Configurable consolidation managers. Historically, consolidation managers focus the placement of VMs with regards to their resource requirements (CPU, memory, I/O, bandwidth, etc.) [6], [21], [25], [26] while reducing the overall energy consumption [24]. These approaches rely on fast ad hoc algorithms to compute the placement.

With the need to provide guarantees that also focus other concerns such as reliability, new ad hoc algorithms, configurable by a fixed set of constraints, have been proposed. Jung et al. consider High-Availability requirements and performance using a utility function [18]. They address server failures, but not load changes. A greedy algorithm computes the number of replicas required for each service composing the affected applications in order to satisfy network latency and performance requirements. Replicas may be placed on different racks or datacenters depending on the desired degree of reliability. Only the VMs of the affected applications are allowed to be relocated to adapt their placement to the new location of the relaunched VMs. Adding new concerns would require revising the reconfiguration algorithm. Experiments are limited to a datacenter of 12 servers. VMware DRS provides the datacenter administrator with 4 constraints similar to `ban`, `fence`, `gather`, and `spread` [12]. Their version of `spread`, however, does not ensure that the VMs will not overlap during reconfiguration. DRS is not meant to be extensible. DRS furthermore does not automatically choose between a re-instantiation or a live migration to relocate a VM depending on the VMs properties nor does it take into account the need for induced relocations. Finally, a cluster managed by DRS cannot exceed 32 servers.

Entropy [17] is at the origin of BtrPlace. It uses CP to place VMs on the minimum number of servers but the scheduling of the actions is computed using an ad-hoc heuristic, which prevents adding constraints such as `spread` that affect the action schedule. Entropy is not designed to be customized. Finally, it always considers all the running VMs when fixing a non-viable configuration, limiting its scalability to a few hundred servers.

Extensible consolidation managers. Some recent approaches propose to provide extensible consolidation managers to integrate placement constraints on demand. Bin et al. [5] also use CP to provide a modular consolidation manager. They provide high-availability by guaranteeing that at all times a certain number of servers will be available that satisfy the VMs resource usage and placement constraints. When the chances of failure for a server are significant, its VMs are migrated to one of the satisfying hosts. Their proposed model does not support constraints related to server state management, action scheduling or the relocation method. Finally, scalability has only been shown for up to 32 servers and 128 VMs.

Some theoretical aspects of BtrPlace were previously investigated [15], with a first prototype that took into account the resources allotted to the VMs, their placement, and their migration. The current paper extends this earlier work by showing the practical extensibility of BtrPlace and its suitability for large datacenters made up of thousands of servers. BtrPlace now infers the most efficient VM relocation method, and makes it possible to control the servers' states, and the resource overallocation. Its extensibility is shown through the addition of 10 new placement constraints related to these concerns. The branching heuristic of BtrPlace, relying on the *filter* optimization, makes it more than 20 times faster for datacenters with 2,500 servers.

6 Conclusion and Future Work

Consolidation of VMs allows multiple applications to share servers within a datacenter. However, modern applications have scalability and high availability requirements that must be taken into account in the consolidation process, and new kinds of constraints on placement are emerging continuously with new uses of datacenters. Reconciling these requirements while allowing server sharing is challenging. We have proposed BtrPlace, a flexible consolidation manager allowing datacenter and application administrators to describe placement constraints in configuration scripts. These scripts are interpreted on the fly to extend a composable reconfiguration algorithm that is used to fix non-viable placements.

The expressivity of BtrPlace has been verified by implementing 14 placement constraints related to resource management, isolation, fault tolerance, and server management. These constraints reproduce, extend but also bring new meaningful restrictions on the VM placement with regards to constraints available in commercial consolidation managers. Each constraint was implemented by an average of 30 lines of Java code. An experienced developer implemented some of the them in half a day, while external developers, without any background in CP, have implemented constraints related to power efficiency [11]. All of these developments did not alter the composability of BtrPlace.

Experiments on a simulated datacenter having 5,000 servers running 30,000 VMs shows BtrPlace can find a viable configuration in less than 3 minutes with placement constraints related to performance and fault tolerance having little impact on the solving time and the quality of the solution. BtrPlace also detects independent placement sub-problems and solves them in parallel without any degradation of the solutions quality. Partitioning the 5,000-server problems into two partitions of 2,500 servers, reduces the total duration of the solving process to 30 seconds. The remaining limitation on the scalability of the approach is then the number of servers available to compute the subproblems simultaneously.

In future work, we want to enable BtrPlace to infer an ideal partitioning by itself. We also plan to enrich BtrPlace through the integration of constraints that can be violated at a penalty. Finally, we want to keep integrating new extensions for the core RP to make BtrPlace able to take into account new resources and concerns.

Availability: All the material related to the reproduction of the scalability experiments is available at https://github.com/fhermeni/benchmarks-tdsc.

Acknowledgments: Experiments presented in this paper were carried out using the Grid'5000 experimental testbed (https://www.grid5000.fr), being developed by INRIA with support from CNRS, RENATER and several universities as well as other funding bodies.

REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. Algorithms for data migration. *Algorithmica*, 57:349–380, 2010.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th SOSP*, pages 164–177, 2003.

[4] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global constraint catalog. Technical Report T2005-08, Swedish Institute of Computer Science, 2005.

[5] E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *31th ICDCS*, June 2011.

[6] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, May 2007.

[7] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. *IEEE/ACM Int. Workshop on Grid Computing*, 2005.

[8] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel. Performance comparison of middleware architectures for generating dynamic web content. In *Middleware*, 2003.

[9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *2nd NSDI*, pages 273–286, 2005.

[10] Dean, Jeff. Designs, lessons and advice from building large distributed systems. In *Keynote of the International Conference on Large-Scale Distributed Systems and Middleware Conference*, 2009.

[11] C. Dupont, G. Giuliani, F. Hermenier, T. Schulze, and A. Somov. An energy aware framework for virtual machine placement in cloud federated data centres. In *3rd IEEE/ACM International Conference on Future Energy Systems, e-energy*, May 2012.

[12] Epping, Duncan and Denneman Frank. *VMware vSphere 4.1 HA and DRS technical deepdive*. CreateSpace, 2010.

[13] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 51–62. ACM, 2009.

[14] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[15] F. Hermenier, S. Demassey, and X. Lorca. Bin repacking scheduling in virtualized datacenters. *Principles and Practice of Constraint Programming–CP 2011*, pages 27–41, 2011.

[16] F. Hermenier, A. Lèbre, and J.-M. Menaud. Cluster-wide context switch of virtualized jobs. In *4th Int. Workshop on Virtualization Technologies in Distributed Computing*, 2010.

[17] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. In *VEE*, 2009.

[18] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Performance and availability aware regeneration for cloud based multitier applications. In *DSN*, 2010.

[19] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, Oct 1985.

[20] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., 2006.

[21] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In *IEEE International Conference on Autonomic Computing*, 2006.

[22] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. Kingfisher: Cost-aware elasticity in the cloud. In *INFOCOM, 2011 Proceedings IEEE*, pages 206–210. IEEE, 2011.

[23] Virtualization penetration rate in the enterprise. Technical report, Veeam Software, 2011.

[24] A. Verma, P. Ahuja, and A. Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08*. Springer-Verlag NY, Inc., 2008.

[25] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75. IEEE, 2011.

[26] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *IEEE 3rd Int. Conf. on Cloud Computing*, 2010.

Fabien Hermenier received the Ph.D degree in 2009 from the University of Nantes. He has been an associate professor at University of Nice Sophia-Antipolis since 2011. His research interests are focused on hosting platforms, virtualization, autonomous computing and resource management. Since 2006, he has been working on virtual machine placement algorithms to cope with the rise of SLAs in hosting platforms.

Julia Lawall received the Ph.D. degree in 1994 from Indiana University. She has been a senior research scientist at INRIA Paris-Rocquencourt since 2011. Previously, she was an Associate Professor at the University of Copenhagen. Her research interests are primarily in the area of improving the quality of infrastructure software, using a variety of approaches including program analysis, program transformation, and the design of domain-specific languages. She is an associate editor of the journals HOSC and SCP, the chair of the GPCE steering committee, and the secretary of IFIP TC2.

Gilles Muller received the Ph.D. degree in 1988 from the University of Rennes I, and the Habilitation a Diriger des Recherches degree in 1997 from the University of Rennes I. After having been a researcher at INRIA and a Professor at the École des Mines de Nantes, he is currently a senior research scientist at INRIA Paris-Rocquencourt. His research interests include the development of new methodologies based on the use of domain-specific languages for the structuring of operating systems. Gilles Muller has been a member of the IEEE since 1995 and was the vice chair of the ACM/SIGOPS from July 2003 to July 2007.