Dynamic Packing with Side Constraints for Datacenter Resource Management

Sophie Demassey, Fabien Hermenier and Vincent Kherbache

Abstract Resource management in datacenters involves assigning virtual machines with changing resource demands to physical machines with changing capacities. Recurrently, the changes invalidate the assignment and the resource manager recomputes it at runtime. The assignment is also subject to changing restrictions expressing a variety of user requirements. The present chapter surveys this application of vector packing – called the VM reassignment problem – with an insight into its dynamic and heterogeneous nature. We advocate flexibility to answer these issues and present BtrPlace, a flexible and scalable heuristic solution based on Constraint Programming.

Key words: datacenter resource management, vector packing, dynamic side constraints, constraint programming

1 Introduction

A datacenter is an infrastructure hosting computing machines. They supply different resources (CPU, RAM, etc.) in limited amount to execute software applications submitted by clients. Thanks to virtualization, a single Physical Machine (PM) can simultaneously run multiple application components, each embedded in a Virtual

Sophie Demassey
MINES ParisTech, PSL - Research University, CMA, CS 10207 rue Claude Daunesse 06904 Sophia Antipolis Cedex, France, e-mail: sophie.demassey@mines-paristech.fr

Fabien Hermenier
University Nice Sophia Antipolis, CNRS, I3S, UMR 7271, France, e-mail: fabien.hermenier@unice.fr

Vincent Kherbache
INRIA, France, e-mail: vincent.kherbache@inria.fr

Machine (VM), if their total demand in each resource does not exceed the PM capacity, i.e. the amount of resource supplied by the PM.

A datacenter is a dynamic system since both demands and capacities vary over time: continuously, VMs are submitted, stopped or resized according to the application needs; continuously, PMs are upgraded, powered on to support load spikes, or halted for maintenance purpose or due to a failure; continuously, execution rules are stipulated by the users of the datacenter – both the operators and the clients – for performance or security purpose. A datacenter is also a market place between the operators, who expect a maximal use of their resources at minimal operation cost, and the clients, who negotiate Quality of Service (QoS) contracts.

The resource manager of a datacenter is responsible for provisioning the submitted workload continuously. It assigns and reassigns VMs to PMs according to the current resource and user requirements so as to optimize QoS, operation costs and resource usage. The problem is a dynamic variant of vector packing with heterogeneous side constraints [17]: *dynamic* since the manager reoptimizes the problem at runtime, and *heterogeneous* since side constraints express a variety of user requirements and preferences.

Datacenters are commonplace nowadays with the advent of cloud computing. As their size keeps growing (to up to thousands of PMs in large IT companies) they necessitate more automation in resource management. Resource managers with advanced optimization abilities are however far from ubiquitous, as the dynamic and heterogeneous nature of the problem remains one major issue.

In this chapter, our first aim is to review this application of vector packing – which we call the VM REASSIGNMENT PROBLEM – with an insight into its two characteristics: dynamicity and heterogeneity. About dynamicity, we further discuss the induced problem of scheduling the reassignment actions. About heterogeneity, we survey some user requirements and preferences met in practical and seminal works. We provide generic formulations of these side constraints which may apply to many other practical applications of packing.

Our second aim is to illustrate the need for *flexibility* in optimization tools to address such characteristics. We present BtrPlace [7] our implementation of a flexible resource manager for virtualized datacenters. BtrPlace relies on Constraint Programming to provide dynamic reassignment and easy customization abilities while yet ensuring performance and scalability.

The chapter is structured as follows: in Section 2, we discuss the concepts of dynamicity, heterogeneity and flexibility in the context of resource management. Section 3 formalizes the core packing problem and variants of the literature, then describes the induced scheduling problem. Section 4 catalogues typical user requirements. Sections 5 and 6 are devoted to BtrPlace and show empirical evaluations. Section 7 presents our conclusions and future research directions.

2 Flexible Resource Management

Datacenter resource management exhibits two facets: *dynamicity* and *heterogeneity*. This section describes how resource managers should accordingly offer *configurability* and *extensibility*. *Flexibility* refers to the combination of these two attributes.

2.1 Configurable managers for dynamic datacenters

The infrastructure and the workload of a datacenter are highly volatile. They change, at variable pace and at variable intensity, as the user activities change and as failures occur. For example, the operators renew PMs in batches every month, they upgrade the PMs overnight, a hardware failure occurs about every day or week [11], the clients submit new applications every hour, and the load of service applications (such as websites) varies in minutes with spikes occurring at morning and off-peaks during weekends. These changes give the VM reassignment problem its dynamic nature and impact it in different ways:

Repair. The problem is not to compute a new assignment but to repair a corrupted one. When changes invalidate the current assignment (e.g, when the new resource demand of a VM suddenly exceeds its current host capacity), the resource manager must compute a new valid assignment, then plan the appropriate reconfiguration actions: powering PMs on and off, launching and migrating VMs either live or off by cloning. These actions affect the performance of the applications during a significant time (e.g., about 10 seconds to halt or migrate live a VM of 1GB RAM [19]). They also incur extra operation costs due to energy consumption and hardware usage. Hence, the resource manager should minimize the effects of the reconfiguration when computing a new assignment.

Reactivity. Since the changes cannot be predicted accurately (e.g., when and where the next hardware failure will occur) and since the applications run in degraded mode while their requirements are violated and during the reconfiguration, a resource manager must (i) operate at runtime, (ii) compute solutions quickly, and (iii) compute fast reconfiguration plans.

Elasticity. In addition to computing an assignment, the resource manager may command the VM and PM states (e.g., launch, halt, sleep) to accommodate the requirements. For example, it may adjust the number of replicas of a service according to the datacenter load and the required degree of fault tolerance. In these settings, the numbers and sizes of the VMs and PMs become new variables of the problem.

Structural changes. Finally, the changes affect the numeric values (the resource requirements) but also the logical constraints (the user requirements) of the problem. From one execution to another, the resource manager is then likely to solve a new optimization problem, not just a new instance of the problem.

Configurability is a required attribute of autonomic resource managers to address structural changes. A configurable resource manager takes as input the current assignment and the new user and resource requirements to merge them into its internal optimization model. If the current assignment violates at least one requirement, it then solves the model. For usability, the manager must offer a high-level interface to specify the new requirements. For reactivity, the internal reformulation of these requirements is expected to be fast. For robustness, the structural changes of the model should not deteriorate the performance of the solution algorithm.

2.2 Extensible managers for heterogeneous datacenters

The infrastructure and the usage make each datacenter unique. The development of an universal resource manager remains utopian. Furthermore, each resource manager must deal with the heterogeneity inherent to its own datacenter.

Infrastructure and workload. The design of a datacenter depends on its function (e.g., for private business, internet service, or cloud computing). The size is a major characteristic as it varies from ten PMs gathered in a room to thousands of PMs geographically distributed. Resource management in such distinct environments refers to distinct problems and requires distinct solutions. Though, any resource manager must be scalable at some extent to support the probable growth of its infrastructure.

Within a datacenter, resources and machines come with a great diversity. Different types of resources are either provided by the PMs (e.g., CPU, RAM, disk storage, network interfaces) or shared by groups of PMs (e.g., licenses). Different PMs supply different types of resources and have different capacities. Furthermore, the PMs are connected through a hierarchical network offering different classes of bandwidth and latency.

Similarly, the workload usually presents a great heterogeneity in sizes and shapes from one application to another. This heterogeneity prevents to rely on symmetry arguments to help solve the packing problem.

User requirements and preferences. Operators – who own or manage the infrastructure – and clients – who submit or use the applications – have multiple needs in terms of resource allocation. Clients expect a reliable QoS to guarantee the optimal execution of their applications by contracting Service Level Agreements (SLAs). SLAs describe low-level metrics (e.g., the resource demand) and logical conditions on the relative assignment of VMs to express different concerns (e.g., grouping communicating VMs on PMs close to each other for performance purpose). Since the client pays for his SLAs and is refunded when violations occur, the operator is willing to enforce QoS while reducing operation costs (e.g., minimizing the number of powered PMs). Operators have strict requirements too, either permanent (e.g., isolating management services on specific PMs for security purpose) or temporary (e.g., freeing PMs to prepare for a maintenance). Hence, a great variety of user re-

quirements and preferences exist. The resource manager must handle a number of them simultaneously at each reassignment step.

Extensibility refers, for a software, to the ease to design and to implement new functionalities. Resource managers release new features to clients on a regular basis. For example, the widely used VMware vSphere and Amazon EC2 were updated to support additional requirements, such as VM-to-PM affinity [13] or dedicated PMs [2]. Extensible resource managers should enable operators to implement desired features. One approach of extensibility is to rely on a modular framework providing an extensible set of primitives to express each feature.

2.3 Related Works

Flexibility is a recent concern in datacenter resource management. Pioneer approaches focused on scalability issues and proposed ad-hoc approximation algorithms ignoring everything but CPU and memory requirements [6, 19, 20, 26, 27]. The increasing energy consumption and the rise of SLAs shifted the goal of resource management to compromise between power saving and QoS guarantee. Ad-hoc partially configurable algorithms have been proposed to support these models (e.g., [13, 21]). The extensibility of these algorithms is however not discussed and the experiments limited to datacenters with less than 50 PMs. In the context of the Roadef/EURO Challenge 2012 [16], Google described a VM reassignment problem with a fixed set of 8 user constraints including 5 violation penalties to minimize. The dataset consisted of synthetic instances up to 5,000 PMs and 20 resources to solve in 5 minutes. The competing algorithms were evaluated with regard to their optimization performance, not their flexibility.

Approaches based on Constraint Programming address extensibility but their experiments are often limited to datacenters of tens PMs (e.g., [5]). BtrPlace and its former version Entropy [18, 19] use Constraint Programming with the aim to address flexibility together with scalability. Currently, BtrPlace is bundled with 16 high-level user requirements but users already developed their own. It also provides a simple configuration language to invoke these side constraints on the fly. BtrPlace computes solutions for simulated instances of 5,000 PMs in less than 1 minute.

3 Problem Statement

This section describes the core VM REASSIGNMENT PROBLEM – without user requirements – as a multi-dimensional vector packing problem. It also presents several objective variants and the induced reconfiguration scheduling problem.

3.1 The VM Reassignment Problem

Definition 1. A datacenter consists of a set $\mathscr{P}$ of PMs (the bins) and a set $\mathscr{R}$ of resources (the dimensions). A workload consists of a set $\mathscr{V}$ of VMs (the items). Each PM $p \in \mathscr{P}$ provides a given amount of each resource $r \in \mathscr{R}$, called its *capacity* and denoted c_{pr}. Each VM $v \in \mathscr{V}$ requires to run a given amount of each resource $r \in \mathscr{R}$, called its *weight*, and denoted w_{vr}. A feasible configuration is an assignment M of the VMs in $\mathscr{V}$ to the PMs in $\mathscr{P}$ that satisfies the *resource requirements*:

$$\sum_{v \in M^{-1}(P)} w_{vr} \leq c_{pr} \quad \forall p \in \mathscr{P}, r \in \mathscr{R}.$$

Given a current *source configuration* $M_0 : \mathscr{V} \to \mathscr{P}$, new capacities $c \in \mathbb{N}^{\mathscr{P} \times \mathscr{R}}$ and new weights $w \in \mathbb{N}^{\mathscr{V} \times \mathscr{R}}$, the VM REASSIGNMENT PROBLEM is to find a *target configuration* $M : \mathscr{V} \to \mathscr{P}$ satisfying the new resource requirements while optimizing a given quantitative *performance goal* $f(M) \in \mathbb{R}$.

3.2 Performance Goals

The performance goal estimates the quality of the target configuration M and of the reconfiguration process to reach M from M_0, in terms of service to the clients and of financial and energy savings for the operators. Performance goals are context-bound but, by contrast to the user requirements, they generally do not vary over time.

A single performance goal is typically integrated with the model as a function to optimize. A weighted sum allows to merge multiple goals as one objective function. However, intensive experiments and practical knowledge are needed to calibrate the weights accurately. An alternative is to bound the function values of a given goal by means of a hard constraint and to reoptimize the problem with progressively tightened bounds. For goals expressing user preferences, the user requirements are modeled as soft constraints that trigger penalty costs possibly proportional to the degree of violation; the objective then is to minimize the sum of the penalties.

Because of the theoretic cost models, the large size of the instances, the computational complexity of the problem, and the allowed solution time, resource managers do not seek optimality when solving the VM reassignment problem in practice.

3.2.1 Scoring the Target Configuration

Workload Consolidation aims at gathering the workload into the minimum number of PMs to power off the unused PMs [6, 19, 26, 27]. With no user requirements, the model coincides with the actual Multi-Dimensional Vector Packing Problem [15]. A more elaborated model, matching the Capacitated Facility Location Problem, estimates the energy consumption through a fixed cost for each active PM

and an execution cost for each pair of VM-PM [12]. Consolidation policies enforce PMs to run at full load. This tends to multiply resource shortages, thus reconfigurations, when the workload is subject to load spikes.

Load Balancing is a performance-oriented policy. It spreads the VMs across the PMs to get a desired load rate on each PM. Such policy can be achieved by minimizing the maximum load over all PMs [1], the sum of the deviations from the desired load rate [24], or the sum of the penalty costs for exceeding a desired safety capacity [16].

SLA protection refers to policies expressing client satisfaction. An example of global satisfaction is to maximize the number of running applications [24]. The problem maps then to the Multiple Knapsack Problem. Individual client demands – such those described in Section 4 – may also be turned into soft constraints then integrated with the objective when their satisfaction is more desired than required [16].

3.2.2 Scoring the Reconfiguration Process

The actions to execute on VMs and PMs to reach the target configuration from the source configuration impact the performance of the datacenter: they provoke downtimes and significant delays, and incur direct operation costs. The reconfiguration score reflects this impact and often dominates the target configuration score in many applications [4, 6, 19, 26, 27]. In fact, performance goals based on reconfiguration scores limit the distance between a source and a target configuration, thus preserve the stability of configuration scores.

Local changes. For instance, the workload consolidation policy yields no energy savings if PMs are turned on and off too frequently. Therefore the performance goal should limit the number of PM state transitions rather than the number of powered PMs. Limiting the number of VM migrations is less trivial. To minimize changes, most works on consolidation [4, 6, 18] and load balancing [14] solve the violations locally – one at a time or altogether – by repairing only a minimal subset of assignments. The decomposition obviously hinders optimality but drastically reduces the problem size.

Migration numbers. An alternative is to focus on minimizing the reconfiguration impact due, in particular, to the VM migrations. In the Load Rebalancing problem [14], the goal is to minimize the number of migrations. In [1], a hard constraint enforces to move less than k VMs. In [16], the maximum number of migrations per application is minimized as well as the weighted sum of migrations between each pair of PMs to simulate network bandwidth conservation.

Action durations. In addition to the number of actions, [16] minimizes the sum of the predefined impacts of the actions. The action duration – including preparation and transfer times – is a relevant indicator of the impact. The duration can itself be evaluated as a function of the type of the action and of the size of the object. Several

works consider this criterion in priority. They minimize either the duration of the whole reconfiguration process [22] or the sum of the completion times [19].

3.3 Scheduling the Reconfiguration Actions

Due to the resource limitations, the reconfiguration actions may have to be scheduled in a specific order. Figure 1a depicts such a situation: since PM p_1 supplies not enough RAM to run VMs v_1 and v_2 together, the resource manager must halt v_1 before starting the migration of v_2 to p_1.

Enabling live migrations makes the resource constraints still harder since a VM consumes resources on both the source and the target PMs during all the time of its live migration. As a result, a cycle of live migrations may cause a deadlock forbidding to reach the target configuration. Figure 1b illustrates this worst case: a cycle occurs between the live migrations of v_1 and v_2 since none of the two available PMs has enough RAM to colocate the two VMs at any time.

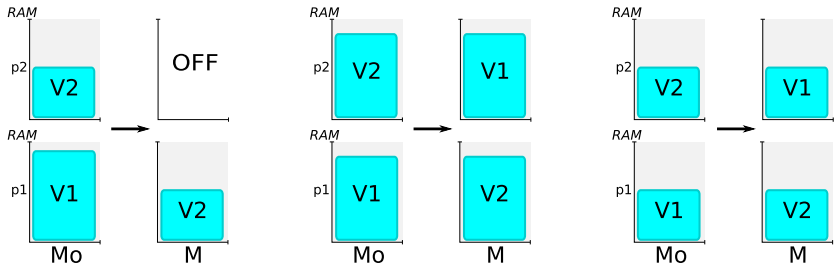


Fig. 1: Impacts of the resource (1a, 1b) and user (1c) requirements on the reconfiguration from M_0 to M: (1a) requires to halt v_1 before migrating v_2, (1b) prevents to migrate live both v_1 and v_2, (1c) prevents to migrate live both v_1 and v_2 if they are in conflict.

A common approach handles the reconfiguration scheduling problem separately after computing the target configuration [19, 27]. It requires to recompute both the reassignment and the scheduling if cycles occur. Furthermore, this two-phases approach disallows to consider the actual impact of the reconfiguration process within the reassignment problem.

As a workaround, [16] tightens up the resource constraints to ensure that all live migrations may happen simultaneously: the total requirement of both the newly assigned VMs and the previously hosted VMs must not exceed a PM capacity. This workaround discards feasible configurations by making permanent the temporary tighter resource requirements induced by live migrations. In addition, it permits to violate user requirements during the reconfiguration. Figure 1c considers the *conflict*

constraint forbidding to colocate the two VMs v_1 and v_2. To enforce this constraint, one of the VMs must be stopped during the migration of the other one, then relaunched on its target PM. The impacts of these service interruptions and delays are not considered in the reassignment model of [16].

Bin et al. [5] consider the continuous satisfaction – including during the reconfiguration – of one user requirement in a particular use case. BtrPlace [7] generalizes this principle to any user requirements: by handling reassignment and scheduling as one global problem, it enforces the continuous satisfaction of the requirements [10] and controls the reconfiguration impact explicitly.

4 User-Defined Side Constraints

In this section, we present a catalog of packing side constraints issued from the literature and from practical user requirements. For each constraint, we discuss its main application contexts, cite some referring works and introduce a mathematical set formulation using notations of Table 1.

Table 1: Notations

$p \in \mathscr{P}$	physical machines (PMs)
$v \in \mathscr{V}$	virtual machines (VMs)
$r \in \mathscr{R}$	resources
$c_{pr} \in \mathbb{N}$	capacity of PM p in resource r
$w_{vr} \in \mathbb{N}$	requirement of VM v in resource r
$M_0 \in \mathscr{P}^{\mathscr{V}}$	source configuration
$M \in \mathscr{P}^{\mathscr{V}}$	target configuration
$P \subseteq \mathscr{P}$	a set of PMs
$C^{\mathscr{P}} \subseteq 2^{\mathscr{P}}$	a set of sets of PMs
$V \subseteq \mathscr{V}$	a set of VMs
$C^{\mathscr{V}} \subseteq 2^{\mathscr{V}}$	a set of sets of VMs

`spread`(V) assigns all the VMs in V onto pairwise distinct PMs [18, 7]. It is named *conflict* in [16], *VM-VM affinity* in [13] and *GroupAntiAffinity* in [23]. *Spread* is relevant to clients for fault-tolerance purpose by avoiding a single point of failure.

$$\text{card}(M(V)) = \text{card}(V).$$

`gather`(V) assigns all the VMs in V onto the same PM [18, 7]. It is named *VM-VM anti-affinity* in VMWare DRS[13]. *Gather* is relevant to clients for performance purpose by improving the intercommunication of a group of VMs.

$$\text{card}(M(V)) = 1.$$

`ban`(V,P) assigns all the VMs in V onto PMs not in P [18, 7]. It is named *VM-PM anti-affinity* in VMWare DRS[13]. *Ban* is relevant to operators for maintenance purpose – by freeing PMs before an upgrade – or for security purpose – by preventing client VMs to run on operator dedicated PMs.

$$M(V) \cap P = \emptyset.$$

`fence`(V,P) assigns all the VMs in V onto PMs in P [18, 7]. It is named *VM-PM affinity* in VMWare DRS[13]. *Fence* is relevant to operators for security purpose by partitioning VMs and PMs according to their compatibility.

$$M(V) \subseteq P.$$

`mostlySpread`(V,n) assigns all the VMs in V to at least $n \in \mathbb{N}$ PMs [7]. It is named *soft VM-VM affinity* in VMWare DRS [13]. *mostlySpread* is a soft version of *spread* when only a minimum number of distinct PMs is required.

$$\text{card}(M(V)) \geq n.$$

`quarantine`(P) prevents the PMs in P to relocate their initial hosted VMs and to host new VMs [13, 18, 7]. *Quarantine* is relevant to operators for security purpose by isolating compromised PMs.

$$\forall p \in P, M^{-1}(p) = M_0^{-1}(p).$$

`among`$(V,C^{\mathscr{P}})$ assigns all the VMs in V onto PMs belonging to a single group of $C^{\mathscr{P}}$ [18, 28, 7]. *Among* is relevant to clients and operators for performance purpose by running strongly communicant VMs on PMs with low network latency.

$$\exists P \in C^{\mathscr{P}}, M(v) \subseteq P.$$

`root`(V) prevents to reassign any VMs in V [7]. It is available as a property in [13]. *Root* is relevant to clients and operators for performance purpose by attaching VMs to some peculiar device.

$$\forall v \in V,\ M(v) = M_0(v).$$

`split`$(C^{\mathscr{V}})$ prevents to collocate VMs belonging to two different groups in $C^{\mathscr{V}}$ (the groups are pairwise disjoint) [7]. It is available as the *dedicated instances* feature in Amazon EC2 [2]. *Split* is relevant to clients for security purpose by isolating groups of VMs from supposed malicious VMs.

$$\forall V_1 \in C^{\mathscr{V}}, \forall V_2 \in C^{\mathscr{V}} \setminus \{V_1\}, M(V_1) \cap M(V_2) = \emptyset.$$

`splitAmong`$(C^{\mathscr{V}},C^{\mathscr{P}})$ assigns each group of VMs belonging to two different groups in $C^{\mathscr{V}}$ to distinct groups in $C^{\mathscr{P}}$ (the groups are pairwise disjoint) [18, 7]. It is a generalization of the *availability zones* in Amazon EC2 [2]. *SplitAmong*

is relevant to clients for fault-tolerance purpose by isolating replicated VMs on dedicated PMs.

$$\forall V_1 \in C^{\mathscr{V}}, \forall V_2 \in C^{\mathscr{V}} \setminus \{V_1\}, \exists P_1, P_2 \in C^{\mathscr{P}}, M(V_1) \subseteq P_1, M(V_2) \subseteq P_2, \text{ and } P_1 \neq P_2.$$

`maxOnline`(P,n) forces at most $n \in \mathbb{N}$ PMs in P to run [7]. *MaxOnline* is relevant to operators for performance purpose by restricting the number of running PMs due to license restrictions or cooling and powering limited capacities [9].

$$\text{card}(P \cap M(\mathscr{V})) \leq n.$$

`capacity`(P,r,c) forces the total amount of resource r consumed on the PMs in P to be lower than $c \in \mathbb{N}$ [7]. *Capacity* is relevant to operators for performance purpose by restricting access to a shared resource, such as the number of Internet Protocol addresses.

$$\sum_{v \in M^{-1}(P)} w_{vr} \leq c.$$

`spreadAmong`$(V, C^{\mathscr{P}})$ assigns the VMs in V to at least $n \in \mathbb{N}$ groups of PMs among $C^{\mathscr{P}}$. It is named *spread* in [16]. *SpreadAmong* is relevant to clients for fault-tolerance purpose.

$$\text{card}\{P \in C^{\mathscr{P}} | M(V) \cap P \neq \emptyset\} \geq n.$$

`dependency`(V_1, V_2, C) given a partition C of the set of the PMs, assigns the VMs in V_1 to elements of C that run at least one VM in V_2 [16]. *Dependency* is relevant to clients for performance purpose.

$$C(M(V_1)) \subseteq C(M(V_2)).$$

5 BtrPlace: a flexible resource manager

In this section, we present BtrPlace, an open source resource manager based on Constraint Programming [7]. BtrPlace is the evolution of the former consolidation manager Entropy [19] with a focus on flexibility [17, 18].

5.1 Global design

For regular users, BtrPlace is a configurable VM reassignment algorithm bundled with 16 placement constraints addressing security, performance, reliability and fault-tolerant concerns. The algorithm takes as input (i) a description of the infrastructure extracted from a monitoring service and (ii) a collection of resource and user requirements declared through an API and configuration scripts [18]. The algo-

rithm first checks if the current infrastructure satisfies all the requirements. If not, it computes a new valid VM-to-PM assignment and a schedule of the reconfiguration actions. For advanced users, BtrPlace is an extensible VM reassignment algorithm where third-party developers can implement and integrate new constraints and extensions. BtrPlace is employed for different usages by companies and in research projects such as the Fit4Green European project [12] which addresses energy efficiency in datacenters.

5.2 Implementing flexibility

The flexibility of BtrPlace results from the composability of its core Constraint Programming model through the use of global constraints [3]. In Constraint Programming, a combinatorial problem is modeled as *variables* taking their values in discrete sets called *domains* and *constraints* that represent the required relations between the variables. Each constraint provides a dedicated algorithm to identify and *filter* values in a variable domain that are inconsistent with regard to the relation and to the other variable domains. A *propagation algorithm* calls the filtering algorithms in turn until no more inconsistencies are detected. If a domain becomes empty the problem is proved to be infeasible. If all domains are singletons then they figure a solution. Otherwise, a decision tree is built. Successively at each node, a variable-value assignment to explore is selected – in a heuristic order called the *branching strategy* – and the propagation algorithm is recalled.

Flexibility is a strength of Constraint Programming compared to other paradigms like Mathematical Programming or SAT solving: A Constraint Programming model decomposes a problem in global constraints which are altogether processed by a generic algorithm. As a constraint may express any logical relation, a user can embed a part of the complexity of his problem in one constraint as soon as he can define a reasonably efficient filtering algorithm for it. Finally, any Constraint Programming solver supplies a – usually extensible – set of fundamental constraints which can be easily invoked through predicates to compose a model. BtrPlace relies on the Java open-source solver Choco [8].

For each call to the reconfiguration algorithm, BtrPlace generates the Constraint Programming model in two phases. The first phase generates the core model including decision variables for the VM assignments, the PM states, the starting times of the reconfiguration actions, and ad-hoc constraints of vector packing and scheduling. The second phase specializes the core model with the resource and user requirements. Each component attaches variables of the core model with new generated variables through some global constraints of the Choco API. The resulting model is solved heuristically by a truncated branch-a-bound.

The extensibility of BtrPlace has yet some limitations. It cannot infer the next VM states, or perform multiple actions on a same element during a reconfiguration. It also historically focuses on hard constraints. Problems that are vastly organized

over soft constraints such as the one formulated for the Roadef challenge [16] could be supported by the framework of BtrPlace but are not currently implemented.

5.3 The optimization algorithm

The optimization problem – assignment and scheduling – is obviously NP-hard and is intractable for medium to large-size datacenters. The BtrPlace algorithm uses two heuristic strategies to accelerate the resolution.

The *filter optimization* limits the set of VMs to reassign [18]. Each constraint uses a dedicated algorithm – similar to the filtering algorithm – to check the viability of the current assignment. On failure, it computes a set of *candidate* VMs to migrate to resolve the conflict. For example, the `spread` constraint checks if the VMs to spread are already on distinct PMs. If not, it selects all collocated VMs. All other VMs are fixed to their initial PM in the model prior to its resolution.

Our second strategy relies on a truncated DFS branch-and-bound with a dedicated branching heuristic. The heuristic first focuses on the assignment variables in decreasing order of criticality: first the running VMs that are no longer hosted on a suitable PM, then all other running VMs, and finally the new VMs to launch. To minimize the number of migrations, the heuristic tries to assign a VM first to its current PM, then to the other possible PMs in random order.

6 Evaluation of BtrPlace

Highly-Available (HA) web applications are typical applications running on datacenters. Their architecture illustrates typical user requirements. They are usually composed of 3 tiers: one deserves static HTTP content, a second one handles the business logic and the last one manages data. To ensure performance and fault-tolerance, each tier is composed of replicated VMs to run on distinct PMs. The replicas of the last tier run databases that must synchronize themselves. To reduce the synchronization latency, they have to run on PMs close together.

To evaluate BtrPlace empirically in a realistic context, we generated workloads made of HA web applications. Each application uses between 6 and 30 VMs with at least 2 VMs per tier. The resource requirements of the VMs are defined according to one of 12 templates; all VMs in a same tier instantiate the same template. Each template defines a demand in RAM ranging from 1 to 3 GB and a maximum CPU usage ranging from 30 to 60 uCPU. The CPU consumption varies at any time randomly between 20% and 90% of its maximum usage. User requirements may be attached to an HA application by means of one *spread* constraint per tier (to model fault-tolerance) and one *among* constraint over the VMs of the third tier (to model synchronization latency).

To evaluate the scalability of BtrPlace, we considered a large datacenter of 5,000 PMs each providing 200 uCPU and 16 GB RAM. To evaluate the impact of the resource usage, we varied the consolidation ratio from 3 to 6 VMs on each PM in accordance with a common observation of real service-oriented datacenters [25]. This amounts to up to 1,700 applications running a total of 30,000 VMs and an overall resource usage varying from 36% to 73%. For each consolidation ratio, we generated 50 instances for different source configurations.

We considered two scenarios of reconfiguration: `LI` simulates Load Increases and `NR` simulates a maintenance for Network Rewiring. In `LI`, the CPU demand of 10% of the applications increases by 30% (capped at 100%): it increases the overall demand by an average of 5%. In `NR`, 5% of the PMs are randomly selected to be powered off for maintenance: it corresponds to the rate of rewiring observed in Google's datacenters at any moment [11].

BtrPlace ran on one core of an Intel Xeon X3440 at 2.53 GHz with 16 GB RAM running Linux 2.6.32-5-amd64 and Sun's JVM 1.8.0. We gave a time limit of 5 minutes and stopped BtrPlace at the first solution.

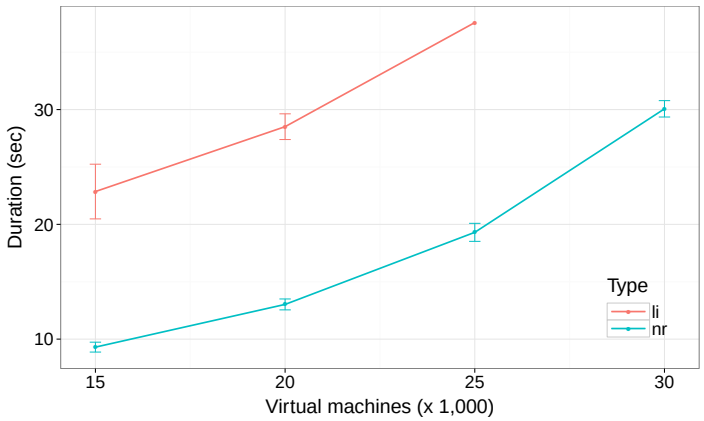
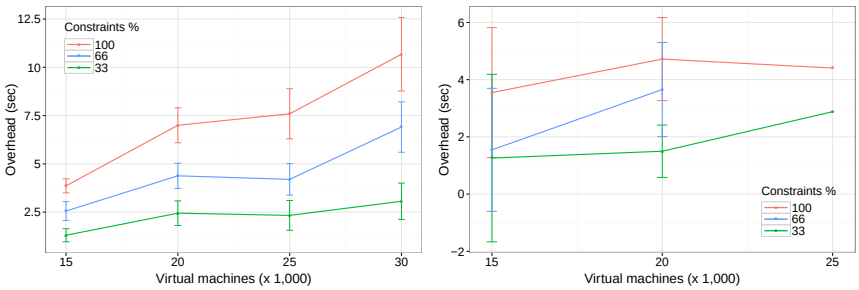


Fig. 2: Solution time according to the number of VMs.

Impact of the number of VMs. Figure 2 shows that the solution time grows with the number of VMs, as expected by the complexity of the problem. However, it never exceeds 30 seconds in the NR case – which is almost the time to halt or to migrate one large VM. Instances of `LI` appear to be much harder as only one of the 50 instances with 25,000 VMs was solved within the 5 allotted minutes and all instances having 30,000 VMs hit the timeout. The difference comes from the *filter optimization* that reduces the problem size more effectively in the `NR` case. In `NR` the algorithm only reassigns VMs that have to be restarted after a failure, while in `LI` all the VMs assigned to overloaded PMs are considered. It amounts to 1,500 VMs and 3,000 VMs respectively on average for the largest instances. The gap of performance is also explained by the intrinsic difficulty of the `LI` case due to

the tighter resource constraints. The number of nodes explored grows exponentially with the consolidation rate.

To address this scalability issue, we envisage three solutions. A first solution is to provide a stronger filtering algorithm of the vector packing constraint since our current implementation is limited on purpose to reduce the memory consumption and to speed up the resolution for large and easy instances. To preserve genericity, we could automatically adapt the filtering level according to the instance characteristics. A second solution is to rely on stronger branching strategies. The strategies must remain instance-independent or, at least, auto-adaptive. The last solution proposed in [18] automatically splits instances into independent sub-problems.



(a) NR (b) LI

Fig. 3: Impact of the user constraints over the solution time

Impact of the user constraints. In the previous experiments, no applications were constrained by the HA requirements (*i.e.* with *spread* and *among* constraints). In Figure 3, we vary the percentage of applications with HA requirements and compute the solution time overhead on the solving process for scenarios `NR` and `LI`.

We observe that the overhead is acceptable in both cases as it never exceeds one third of the total solution time. In the worst case, the average overhead is 11 sec. (34%) in `NR` and only 4 sec (11%) in `LI`. This low overhead demonstrates that the resource constraints are dominant. We also observe that with 25,000 VMs, the total solution time decreases when there is up to 66% of the constraints in the `NR` case and 100% in the `LI` case. This phase transition reveals that the reduction of the search space due to the side constraints may compensate the extra computation time.

7 Conclusion and Future Works

The resource manager has the critical task to efficiently deploy the client applications throughout a datacenter. This involves to assign VMs to PMs and to revise the assignment recurrently as the environment changes. This optimization problem

matches the multi-dimensional vector packing problem with various objectives and side constraints depending on the context. For the past years, many companies and researchers have proposed meaningful solutions for their own context.

In this chapter we advocated an unifying approach that would be able to embrace all these specificities. We emphasized the dynamic and heterogeneous nature of resource management and proposed flexibility as a solution. We assessed this approach through BtrPlace, a flexible and scalable solution based on Constraint Programming. While fully generic optimization algorithms may not always be faster than *ad hoc* solutions, our experiments showed that BtrPlace is effective to manage thousands of highly-available web applications running on thousands of PMs.

In future works, we plan to keep improving BtrPlace in terms of performance and flexibility. Regarding performance, we will develop enhanced algorithmic components (partitioning, filtering, branching) to address the scalability issue when solving both large and difficult instances. Regarding configurability, our next step is to make the algorithm auto-adaptive. Similarly to the model, the solver will automatically invoke the right algorithmic components with respect to the instance characteristics. Finally regarding extensibility, we plan to add support for soft constraints – to manage user preferences in addition to user requirements – and for network concerns – by modeling new elements like topology, bandwidth and latency. One main challenge will be to multiply the case studies to further assess the gain of flexibility.

Availability

BtrPlace is available online under the terms of the LGPL license [7]. Instances used in the experiments are available at https://github.com/btrplace/workloads-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. Aggarwal, G., Motwani, R., Zhu, A.: The load rebalancing problem. In: 15th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '03, pp. 258–265 (2003)
2. Amazon EC2. http://aws.amazon.com/ec2/
3. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog, 2nd edition. Tech. rep., Swedish Institute of Computer Science (2010). http://sofdem.github.io/gccat/
4. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. **28**(5), 755–768 (2012)

5. Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E., Moatti, Y., Lorenz, D.: Guaranteeing High Availability Goals for Virtual Machine Placement. In: International Conference on Distributed Computing Systems (2011)
6. Bobroff, N., Kochut, A., Beaty, K.: Dynamic placement of virtual machines for managing sla violations. In: 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07), pp. 119–128 (2007)
7. BtrPlace: An Open-Source flexible virtual machine scheduler. http://www.btrplace.org/
8. Choco: an open source Java constraint programming library. http://choco.emn.fr
9. Citrix store. http://store.citrix.com
10. Dang, H.T., Hermenier, F.: Higher sla satisfaction in datacenters with continuous vm placement constraints. In: 9th ACM Workshop on Hot Topics in Dependable Systems, HotDep '13, pp. 1:1–1:6 (2013)
11. Dean, Jeff: Designs, lessons and advice from building large distributed systems. In: Keynote of the International Conference on Large-Scale Distributed Systems and Middleware (2009)
12. Dupont, C., Hermenier, F., Schulze, T., Basmadjian, R., Somov, A., Giuliani, G.: Plug4green: A flexible energy-aware VM manager to fit data centre particularities. Ad Hoc Networks (2014)
13. Epping, D., Frank, D.: VMware vSphere 4.1 HA and DRS technical deepdive (2010)
14. Fukunaga, A.S.: Search spaces for min-perturbation repair. In: 15th International Conference on Principles and Practice of Constraint Programming, CP'09, pp. 383–390 (2009)
15. Garey, M.R., Graham, R.L., Johnson, D.S., Yao, A.C.C.: Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory, Series A **21**(3), 257–298 (1976)
16. Roadef/Euro Challenge 2012: Machine Reassignment. http://challenge.roadef.org/2012/
17. Hermenier, F., Demassey, S., Lorca, X.: Bin repacking scheduling in virtualized datacenters. In: Principles and Practice of Constraint Programming, CP'11, pp. 27–41. Springer Berlin Heidelberg (2011)
18. Hermenier, F., Lawall, J., Muller, G.: Btrplace: A flexible consolidation manager for highly available applications. IEEE Transactions on dependable and Secure Computing p. 1 (2013)
19. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a Consolidation Manager for Clusters. In: ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM (2009)
20. Hermenier, F., Loriant, N., Menaud, J.M.: Power management in grid computing with xen. In: International Conference on Frontiers of High Performance Computing and Networking, ISPA'06, pp. 407–416 (2006)
21. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: 30th IEEE International Conference on Distributed Computing Systems, ICDCS '10, pp. 62–73 (2010)
22. Nus, A., Raz, D.: Migration plans with minimum overall migration time. In: Network Operations and Management Symposium (NOMS), 2014 IEEE, pp. 1–9 (2014)
23. Openstack cloud software. http://www.openstack.org/
24. Tang, C., Steinder, M., Spreitzer, M., Pacifici, G.: A scalable application placement controller for enterprise data centers. In: 16th ACM International Conference on World Wide Web, pp. 331–340 (2007)
25. Virtualization penetration rate in the enterprise. Tech. rep., Veeam Software (2011)
26. Verma, A., Ahuja, P., Neogi, A.: pmapper: Power and migration cost aware application placement in virtualized systems. In: Middleware 2008, *Lecture Notes in Computer Science*, vol. 5346 (2008)
27. Wood, T., Tarasuk-Levin, G., Shenoy, P., Desnoyers, P., Cecchet, E., Corner, M.D.: Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In: ACM SIGPLAN/SIGOPS International Conference on Virtual execution environments, pp. 31–40 (2009)
28. Zheng, J., Ng, T.S.E., Sripanidkulchai, K., Liu, Z.: COMMA: Coordinating the Migration of Multi-tier Applications. In: 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '14 (2014)