Bin Repacking Scheduling in Virtualized Datacenters

Fabien Hermenier[1], Sophie Demassey[2], and Xavier Lorca[2]

[1] University of Utah, School of Computing
fhermeni@cs.utah.edu,
[2] TASC project, Mines Nantes-INRIA, LINA CNRS UMR 6241,
firstname.lastname@mines-nantes.fr

Abstract. A datacenter can be viewed as a dynamic bin packing system where servers host applications with varying resource requirements and varying relative placement constraints. When those needs are no longer satisfied, the system has to be reconfigured. Virtualization allows to distribute applications into Virtual Machines (VMs) to ease their manipulation. In particular, a VM can be freely migrated without disrupting its service, temporarily consuming resources both on its origin and destination.
We introduce the Bin Repacking Scheduling Problem in this context. This problem is to find a final packing and to schedule the transitions from a given initial packing, accordingly to new resource and placement requirements, while minimizing the average transition completion time. Our CP-based approach is implemented into Entropy, an autonomous VM manager which detects reconfiguration needs, generates and solves the CP model, then applies the computed decision. CP provides the awaited flexibility to handle heterogeneous placement constraints and the ability to manage large datacenters with up to 2,000 servers and 10,000 VMs.

1 Introduction

A datacenter is a hosting platform of hundreds of interconnected servers. To make this infrastructure costly effective, it should be able to host simultaneously a large range of client applications. Virtualization eases the deployment of the applications. An application is distributed into Virtual Machines (VMs) which can be colocated on any servers, and dynamically manipulated under different kinds of actions, including live migration between hosts.

The deployment of the applications is constrained by the finite capacity of the servers in shared resources like CPU and memory. In addition, clients and system administrators have specific expectations with regards to the relative placement of the VMs on the servers. For instance, a client may require the replicas of his application to be continuously hosted on distinct servers to ensure fault tolerance, or an administrator may require to isolate a group of servers for maintenance purpose.

A system *configuration* is an assignment of the VMs to the servers which satisfies both the resource and placement requirements. Over time, the system evolves: placement requirements vary, servers are intentionally or unexpectedly halted or restarted, VMs are launched or killed and their resource requirements change. When the current configuration is no longer viable, a new configuration has to be computed, and the transition actions to apply to reach the new configuration have to be planned to ensure their feasibility.

The *reconfiguration problem* is then made of a packing and a scheduling subproblems, both being subject to resource and placement constraints. The problem may have no solution, either because no feasible final configuration exists, or because reaching such a configuration induces an unsolvable cycle of transitions. Furthermore, a reconfiguration has an impact on the running applications. Thus the objective is to get a reconfiguration plan with minimum average transition completion time. Although this objective affects the scheduling part only, it is also conditioned by the packing to reach.

The reconfiguration problem is clearly intractable in theory. In practice, the automated VM manager of the datacenter needs to periodically solve online large-sized instances. For scalability reasons, incomplete optimization is then required and a tradeoff has to be made between the computation time of the reconfiguration plan and its execution time. In a previous work [6] on a reconfiguration problem without side constraints, we shown that despite a fast computation time, a standard greedy algorithm tends to compute reconfiguration plans with large execution durations. On the opposite, our solution, partially based on CP, computed faster plans in an extra time that was drastically lesser than the execution duration gain. In this paper, we follow this viewpoint: VM managers would benefit from embedding smarter reconfiguration decision modules. By smarter, we mean able to compute high-quality decisions even if an additional, still acceptable, solving time is required. We mean also flexible and generic enough to handle the various user requirements, which are traditionally not considered. For these reasons, we first adopt a centralized solution approach, by contrast to cheaper distributed approaches which only apply local changes. Although the problem induces a natural decomposition into two subproblems, we solve them conjointly as they both contribute to get a reliable and fast reconfiguration plan. Finally, we rely on Constraint Programming to easily handle the problem as a whole, including any combinations of user requirements.

In this paper, we first formalize the general reconfiguration problem and discuss its complexity (Section 2). We describe the specifications of the practical problem of automated VM management and show how CP fulfills them (Section 3). We then present the CP model, the search strategy, and the two resolution modes we developed for this problem (Section 4). All these elements are pre-implemented into the autonomous VM manager Entropy [6]. Experiments on realistic workloads show that our implementation solves problems involving 10,000 VMs on 2,000 servers with 900 side constraints in less than 5 minutes (Section 5). Last, we review the literature on process and data migration, and

show how our general model fits many of these related problems (Section 6). Our conclusions and perspectives are provided in Section 7.

2 Core Problem Statement

Without side constraints, a configuration is a standard packing of items with arbitrary heights (VMs) to multidimensional bins with arbitrary capacities (servers). A reconfiguration plan is a schedule of the transition actions applied to the VMs, subject to the same resource limitations. The specificity of this scheduling problem comes from the occupation of the resources by each VM: on its initial host before and during the transition, and on its final host during and after the transition. The reconfiguration problem can be dissociated from the context of VM management. To our knowledge, no such formalization has formerly been proposed. Hereafter it is referred to as the Bin Repacking Scheduling Problem.

2.1 The Repacking and Scheduling Problem

Consider a 2-states (initial/final) dynamic system which consists of a set $\mathcal{R}$ of p-dimensional bins with static capacities $B_r \in \mathbb{N}^p$, for all $r \in \mathcal{R}$, and a set $\mathcal{J}$ of items with dynamic initial $b_j^o \in \mathbb{N}^p$ and final $b_j^f \in \mathbb{N}^p$ heights, for all $j \in \mathcal{J}$. The initial state of the system is known and defined as an assignment $s_o : \mathcal{J} \to \mathcal{R}$ satisfying $\sum_{j \in s_o^{-1}(r)} b_j^o \leq B_r$ for each bin $r \in \mathcal{R}$.[3] The system state changes by applying a transition action to each item $j \in \mathcal{J}$. The restricted set of allowed transitions is given as a table $\Delta_j \subseteq \mathcal{T} \times \mathcal{R}$, where each element $\delta = (\tau, r)$ indicates that a transition of type $\tau \in \mathcal{T}$ can be applied to item $j \in \mathcal{J}$ to reassign it from bin $s_o(j)$ to bin r. With any transition $\delta \in \Delta_j$ are associated a duration $d_\delta \in \mathbb{N}$ and a weight $w_\delta \in \mathbb{N}$.

Definition 1. *The* Bin Repacking Scheduling Problem (BRSP) *is to associate with each item $j \in \mathcal{J}$, a transition $\delta(j) = (\tau(j), s_f(j)) \in \Delta_j$ and a time $t_j \in \mathbb{N}$ to start this transition, such that the bin capacities are satisfied at any time*

$$\sum_{\substack{j \in s_o^{-1}(r) \\ t < t_j + d_{\delta(j)}}} b_j^o \; + \sum_{\substack{j \in s_f^{-1}(r) \\ t \geq t_j}} b_j^f \leq B_r, \quad \forall r \in \mathcal{R}, \forall t \geq 0, \tag{1}$$

and the weighted sum of the completion times is minimized

$$\sum_{j \in \mathcal{J}} w_{\delta(j)}(t_j + d_{\delta(j)}), \tag{2}$$

or to prove that no such feasible packing or scheduling exists.

We deliberately present a first conceptual model as the notion of allowed transition action is context-dependent. In the context of VM management, we consider

[3] $s^{-1}(r) \subseteq \mathcal{J}$ denotes the preimage of $\{r\} \subset \mathcal{R}$ under function s from $\mathcal{J}$ to $\mathcal{R}$.

3 groups of items: items $j \in \mathcal{J}_S$ have to be suspended ($b_j^o > 0$, $b_j^f = 0$), items $j \in \mathcal{J}_L$ have to be launched ($b_j^o = 0$, $b_j^f > 0$), items $j \in \mathcal{J}_A$ have to be let active ($b_j^o > 0$, $b_j^f > 0$). To each group corresponds one transition table:

$$\Delta_j = \begin{cases} \{(S, r) \mid r \in \mathcal{R}\} & \forall j \in \mathcal{J}_S \\ \{(L, r) \mid r \in \mathcal{R}\} & \forall j \in \mathcal{J}_L \\ \{(F, s_o(j))\} \cup \{(M, r) \mid r \in \mathcal{R} \setminus \{s_o(j)\}\} & \forall j \in \mathcal{J}_A. \end{cases}$$

The transition types $\mathcal{T} = \{S, L, M, U\}$ stand for *Suspend, Launch, Migrate, Unmoved*, respectively. The duration and the weight of a transition are 0 if it is of type U, and the duration is positive and the weight is 1 otherwise. Let VM Repacking Scheduling Problem (VRSP) refer to this instance of the BRSP. This model totally fits our application: VMs can be suspended (S) or resumed (L), either on their current server or on another, incurring different costs in these two cases. Running VMs can also either stay on their origin server (U) or migrate live to another server (M). In the first case, the transition is immediate ($d_U = 0$), even if the VM resource requirements change, and it does not alter the VM service ($w_U = 0$). Finally, the action of turning a server off or on can be modeled by introducing a dummy VM, respectively to be launched or suspended, statically assigned to the server, and occupying its entire resources.

In the VRSP, the transition typecast is determined by the item j itself, its origin $s_o(j)$ and destination $s_f(j)$ bins. Hence, determining a set of transitions δ_j comes to compute a Multidimensional Bin Packing. This problem is NP-complete in the strong sense [5] even in the one-dimensional case ($p = 1$). In turn, determining the times t_j yields to a particular scheduling problem.

2.2 The Scheduling Subproblem

Definition 2. *Given a final packing $s_f : \mathcal{J} \to \mathcal{R}$ such that $\sum_{j \in s_f^{-1}(r)} b_j^f \leq B_r$, $\forall r \in \mathcal{R}$, and a transition $\delta(j) \in \Delta_j$ for each item $j \in \mathcal{J}$, the* Repacking Transition Scheduling Problem (RTSP) *is to schedule all the transitions such that the resource constraints* (1) *are satisfied and the weighted sum of the completion times* (2) *is minimized, or to prove that no such schedule exists.*

This can be viewed as a Resource Constrained Scheduling Problem [5] with no-wait, variable durations and consumer/producer tasks : to each item $j \in \mathcal{J}$ correspond two operations, O_j occupying b_j^o resource units on $s_o(j)$ in time interval $[0, t_j + d_j)$ and F_j occupying b_j^f resource units on $s_f(j)$ in $[t_j, \bar{H})$, where $\bar{H}$ denotes any large enough scheduling horizon. A decision variant of this problem, with unit durations $d_j = 1$ and constant requirements $b_j^o = b_j^f$, has previously been studied by Sirdey et al. [8] and referred to as Zero-Impact Process Move Program. It asks whether a total order exists over the set of transitions. As durations are unit, this is equivalent to find a timed schedule. In [8], this problem is proved to be NP-hard in the strong sense. We give below a sketch of the proof.

Proposition 1. *The decision variant of* RTSP *is NP-hard in the strong sense, even with 2 one-dimensional bins, unit durations and constant requirements.*

Proof. Consider an instance of 3-PARTITION [5], made of a bound $W \in \mathbb{N}$ and a set A of $3m$ elements of sizes $W/4 < w_a < W/2$ for all $a \in A$ such that $\sum_{a \in A} w_a = mW$. This reduces to an instance of RTSP with two one-dimensional bins $\mathcal{R} = \{r_1, r_2\}$ each of capacity mW and two sets of items: $\mathcal{J}_1$ composed of $3m$ items of height w_j migrating from r_1 to r_2, and $\mathcal{J}_2$ composed of $k-1$ items of height W and migrating from r_2 to r_1. Then finding a partition of A in m sets, each of size W, is equivalent to find a migration plan transferring a height W of resource, alternatively from r_1 to r_2 and from r_2 to r_1. □

3 An Automated VM Manager Relying on CP

The VRSP models a reconfiguration problem centered on the resource requirements. In practice, a VM manager should also deal with user placement requirements. This section first presents, as a proof of concept, 4 typical placement constraints. It then describes the concrete specifications of a VM manager and why CP is suitable. It finally presents the VM manager Entropy that relies on CP for modeling and solving, on-the-fly, instances of a specialized VRSP.

3.1 Side Placement Constraints

A side placement constraint restricts the assignment of given VMs to given servers, or the relative assignments of sets of VMs. Some restrictions are required by the system administrators for management purposes; others are required by the clients for the good execution of their applications. The 4 examples below are representative of concrete standard requirements.

Ban. To perform a hardware or a software maintenance on a server, a system administrator has first to migrate the hosted VMs to other servers. More generally, administrators and clients may want to disallow a given set of VMs to be hosted by a given set of servers. We refer to this constraint as *ban*.

Spread. Highly-available applications use replication to achieve tolerance to hardware failures. To be fully effective, the VMs running the replicas must, at any time, be hosted on distinct servers. We refer to this constraint as *spread*. Figure 1 depicts an instance of VRSP with two VMs subject to a *spread* constraint. As its resource requirements increase, `VM1` has to migrate to server `N2`. *Spread* enforces to delay this action after the migration of `VM2` to `N3` is completed.

Lonely. Using a denial-of-service, a VM may over-use the CPU and memory resources of its host and then impact the colocated VMs or crash the host. A solution is to make critical application VMs to be hosted on servers on their own. Typically, a system administrator separates the service VMs that manage the datacenter from the client VMs. We refer to this constraint as *lonely*.

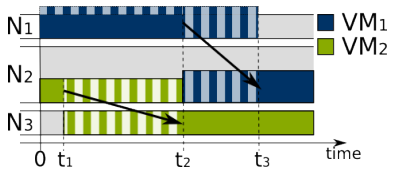

Fig. 1: `spread` enforces `VM1` and `VM2` to always be hosted on distinct servers.

Capacity. A system administrator controls how shared resources are distributed among the VMs. For instance, each VM reachable from outside the datacenter requires a public IP address. As the pool of public IPs is limited, the number of simultaneous running VMs is restricted to the size of this pool. We refer to as *capacity* the constraint that limits the maximum number of VMs colocated on a given set of servers.

3.2 Constraint Programming for VM management

The Autonomous VM Manager of a datacenter relies on a periodic or event-driven control loop composed of four modules: *monitoring* retrieves the current system configuration, *provisioning* predicts the future requirements, *plan* computes the reconfiguration plan, and *execution* performs the physical reconfiguration. The plan module gathers the informations of the monitoring and provisioning modules, adapts the solution algorithm, and runs it. The specifications for an efficient plan module are as follows. First, the algorithm should scale up to the size of the datacenter. Second, as the applications run in degraded mode until the configuration becomes viable, computing a solution should be fast and the reconfiguration durations of the applications should be short. Third the algorithm does not need to ensure optimality, but it is strongly required to be flexible. Indeed, it must be dynamically adaptable to handle different types of side constraints and to deal with any combinations of them. Last, virtualized datacenters exist for a short while, but they spread rapidly and new needs emerge with new usages. As a consequence, a VM manager should be extensible to take into consideration future needs.

Constraint Programming is known as a suitable solution for packing and scheduling problems. We claim that CP actually offers unique capabilities to deal with the practical reconfiguration problem considered here. First, modeling with global constraints eases the specification of new side placement constraints. Second, the propagation engine ensures the automatic composability needed to handle the packing and scheduling problems together with extra placement constraints. Finally, the framework of tree search can easily be specialized in most CP solvers with pre-implemented or ad-hoc variable and value ordering heuristics. Such framework is thus convenient to quickly develop and test complete or local search strategies. The search strategy matches the optimization objective, while the CP propagation engine enforces the feasibility part of the problem.

The statement of the `lonely` constraint illustrates well the flexibility of our approach. This constraint was specified after Amazon EC2 described this new

feature in march 2011.[4] Its whole implementation in Entropy, from the selection of the appropriate global constraint to the tests, has taken only 3 hours, and its model, relying on a global constraint already available in the CP solver Choco (see next Section), is about 50 lines of code. We did not have to modify our heuristics to take this new constraint into account. The same holds true for the 3 other placement constraints described above. Obviously, the expressivity and flexibility of CP have their limits, yet we have not reached them in our current application.

Entropy is an open-source autonomous VM manager.[5] The specificity of Entropy lies in its plan module based on the CP Solver Choco[6] and in its configuration script language for its specialization. The scripts allow administrators and clients to each describe a datacenter and an application, respectively, while focusing on their primary concerns: the administrator manages its servers without any knowledge of the hosted applications, while a client specifies its placement requirements without knowledge of the infrastructure or the other hosted applications. Listing 2a illustrates the description of a 3-tiers highly-available application. A tier (`$T1`, `$T2`, or `$T3`) is composed of several VMs, each running a replica of a same service. For fault tolerance, a *spread* constraint enforces all the VMs of each tier to be placed on distinct servers. To improve the application isolation, a *lonely* constraint enforces all the VMs to be placed on servers on their own. Listing 2b illustrates administrator needs. It describes a datacenter made of 3 racks (`$R1`, `$R2`, `$R3`) of 50 servers each. A maximum of 100 hosted VMs per rack is enforced by 3 *capacity* constraints. Last, all VMs are disallowed to be placed on server `N101` in order to prepare a maintenance.

```
$T1 = VM[1..5];
$T2 = VM[6..15];
$T3 = {VM17, VM21, VM22};
for $t in $T[1..3] {
    spread($t);
}
lonely($T1 + $T2 + $T3);
```

(a) Description of a 3-tiers HA application.

```
$R1=N[1..50];
$R2=N[51..100];
$R3=N[101..150];
for $r in $R[1..3] {
    capacity($r, 100);
}
ban($ALL_VMS, N101);
```

(b) Description of a datacenter.

Fig. 2: Sample configuration scripts provided by clients or administrators.

Given the current configuration retrieved by the monitoring module and the future resource requirements estimated by the provisioning module, the plan module first generates a Choco model of the corresponding VRSP instance. The configuration scripts are then interpreted and the placement constraints are added in turn to the model. If the current configuration is consistent with this model, then Entropy restarts the control loop. Otherwise, the model is optimized

[4] https://aws.amazon.com/dedicated-instances/
[5] http://entropy.gforge.inria.fr
[6] http://choco.emn.fr

for a limited time and the best solution found, if exists, is sent to the execution module in charge to apply the reconfiguration plan.

4 Elements of Solution

This section presents the CP model including the four examples of placement constraints, the search strategy dedicated to incomplete optimization and the two modes of resolution currently implemented in Entropy. The model relies on several standard constraints mentioned in the Global Constraint Catalog [2]; details on these constraints can be found in this reference.

4.1 Modeling the Core Problem

The end of the schedule is the first time the final configuration is reached. In our model, it is represented by a domain variable H, defined on the integer interval $[0, \bar{H}]$, $\bar{H}$ being the given horizon. In order to properly represent a schedule, we first introduce the notion of task variable:

Definition 3. *A* task variable *is a compound object made of integer variables* $T = \langle T^s, T^e, T^r, T^{b1}, T^{b2}, \ldots, T^{bp} \rangle$ *denoting respectively:* T^s *and* T^e *the starting and the ending times of the task,* T^r *the resource the task is assigned to, and* $T^{b1}, \ldots, T^{bp}$ *the heights of the task in the* p *dimensions.*

Producer/consumer tasks. Each VM $j \in \mathcal{J}$ is modeled by two multidimensional task variables representing the occupation of the initial server (O_j) and of the final server (F_j). Such a representation is a variant of the producer-consumer model [7] with no negative stock: O_j *produces* resources at the transition completion time, while F_j *consumes* resources from the transition start time. The two task variables associated with each VM $j \in \mathcal{J}$ are formally defined by:

- $O_j = \langle 0, O_j^e, s_o(j), b_j^{o1}, \ldots, b_j^{op} \rangle$ where only O_j^e is a variable defined on $[0, \bar{H}]$. It means j occupies heights b_j^o on initial server $s_o(j)$ from time 0 to O_j^e.
- $F_j = \langle F_j^s, H, F_j^r, b_j^{f1}, \ldots, b_j^{fp} \rangle$, where F_j^s is a variable defined on $[0, \bar{H}]$, and F_j^r is a discrete variable defined on $\mathcal{R}$. It means j occupies heights b_j^f on final server F_j^r from time F_J^s to the end of the schedule H.

Transition types and no-wait. The tasks associated with a VM $j \in \mathcal{J}$ are subject to a *precedence relation with no-wait*, $O_j^e - F_j^s = d_{\delta(j)}$, which depends on the applied transition action $\delta(j)$. In order to model transition $\delta(j)$, we consider a variable X_j defined on $\mathcal{T}$ denoting the transition type, and a variable W_j, defined on $\mathbb{N}$ denoting the transition weight. Then, the different variables associated with a VM can be related by one table constraint:

$$(X_j, F_j^r, O_j^e - F_j^s, W_j) \in \{(\tau, r, d_\delta, w_\delta) \mid \delta = (\tau, r) \in \Delta_j\}, \quad \forall j \in \mathcal{J}.$$

Resource constraints. The resource constraints can be modeled on each dimension by one `cumulatives` constraint as follows:

$$\texttt{cumulatives}(\langle O_j, F_j \mid j \in \mathcal{J}\rangle,\ \mathcal{R},\ \leq,\ k), \quad \forall k \in \{1, \ldots, p\}.$$

This signature is slightly different from the original one, as it specifies the dimension k to constrain. The filtering of `cumulatives` runs in $O(|\mathcal{R}|.|\mathcal{J}|^2)$. Actually, as this constraint was not available in Choco, we developed our own version specialized to producer/consumer tasks, running with the same time complexity.

Redundant constraints. A transition typed as *Unmoved* has no duration and no cost. It can then be scheduled at any time freeing maximum resources. Formally, for each solution of VRSP, there exists a solution of equal or least cost where an *Unmoved* transition is scheduled at time 0, if the VM requirements decrease, or at time H, if they increase. The property remains true when adding any side placement constraints, as those considered VMs keep precisely the same placement. The property only applies to VMs which requirements vary uniformly in all dimensions, which is usually the case in practice.

$$\begin{aligned} &X_j = U \ \Rightarrow\ O^e_j = F^s_j = 0, && \forall j \in \mathcal{J}_A \mid b^o_j \geq b^f_j, \\ &X_j = U \ \Rightarrow\ O^e_j = F^s_j = H, && \forall j \in \mathcal{J}_A \mid b^o_j < b^f_j. \end{aligned}$$

4.2 Modeling the Side Constraints

Ban. The model of *ban* is straightforward as it relies on a simple domain reduction of the final assignment variables. For any subset of VMs $J \subseteq \mathcal{J}$, and any subset of servers $R \subseteq \mathcal{R}$, constraint $ban(J, R)$ is modeled by:

$$F^r_j \neq r, \quad \forall j \in J,\ \forall r \in R.$$

Spread. Despite appearances, the model of *spread* cannot rely on `disjunctives` constraints as the specified VMs are possibly hosted by a same server in the initial configuration. An alternative is to ensure that the VMs are on distinct servers in the final configuration; then, on each server, to ensure that the arrival of a VM is delayed after all other involved VMs left. For any subset of VMs $J \subseteq \mathcal{J}$, constraint $spread(J)$ is modeled by:

$$\begin{cases} \texttt{allDifferent}(\langle F^r_j \mid j \in J\rangle), \\ F^r_i = s_o(j) \ \Rightarrow O^e_i \leq F^s_j, \quad \forall i, j \in J, i \neq j. \end{cases}$$

Lonely. The model of *lonely* relies on one *disjoint* constraint enforcing the set of servers hosting the specified VMs to be disjoint from the set of servers hosting the remaining VMs. For any subset of VMs $J \subseteq \mathcal{J}$, $lonely(J)$ is modeled by:

$$\texttt{disjoint}(\langle F^r_j \mid j \in J\rangle, \langle F^r_j \mid j \notin J\rangle).$$

Capacity. The model of *capacity* relies on a redundant set model for the VRSP. A set variable, associated with each server, indicates the hosted VMs. The constraint bounds the sum of the set cardinalities over the specified servers. For any subset of servers $R \subseteq \mathcal{R}$ and value $n \in \mathbb{N}$, $capacity(R, n)$ is modeled by:

$$\begin{cases} \sum_{r \in R} \texttt{card}(V_r) \leq n, & \\ j \in V_r \iff F_j^r = r, & \forall r \in \mathcal{R},\ \forall j \in \mathcal{J}, \\ V_r \subseteq \mathcal{J}, & \forall r \in \mathcal{R}. \end{cases}$$

4.3 Solving the VRSP

Dedicated Search Strategy. Entropy solves the CP model using a time-truncated branch-and-bound. The search strategy is conceived to descend quickly towards a local optimum, by following the natural decomposition of the problem. First, it focuses on the final packing and instantiates the assignment variables $\langle F_j^r \rangle_{j \in \mathcal{J}}$. Starting with the VMs whose placement in the initial configuration violates a resource or a placement constraint, the heuristic selects the VMs in order of decreasing memory requirements and attempts at placing them to their initial host first, then to another server selected in a worst-fit fashion. Once the final packing is instantiated, the tasks $\langle F_j \rangle_{j \in \mathcal{J}}$ are started as early as possible, in turn, starting from the tasks which are entering a server with no leaving transition.

A Repair Approach. We experimented two modes of resolution: either starting from scratch or from a partial solution. In the *rebuild mode*, all VMs are allowed to migrate, contrary to the *repair mode* where some candidates are *a priori* fixed to their current location. The repair mode may drastically reduce the size of the model – and then speed and scale up the solution process – if a maximum number of candidates is fixed. On the other hand, the pre-packing should be loose enough to ensure a solution to exist. The issue here is to build a feasible and reasonable-sized partial solution. For this, we compute the intersection of the candidate sets returned by simple heuristics that come with each resource and side constraint.

5 Evaluation

In this section, we evaluate the solving abilities of Entropy on realistic workloads. The critical parameters we evaluate are the consolidation ratio, the size of the datacenter, and the side constraints.

For these experiments, we simulate a datacenter composed of racks with 50 servers each. Each server provides 80 GB RAM and 150 uCPU (an abstract unit to establish the computing capacity of a server). This infrastructure hosts 3-tiers applications, each composed of 20 VMs. The VMs are sized according

to the standards defined by Amazon EC2[7]. The first and the second tiers are composed of 5 and 10 VMs, respectively. Each VM uses 7.5 GB RAM and at most 4 uCPU (*large instances* in the EC2 terminology). The third tier is composed of 5 VMs, each using 17.1 GB RAM and at most 6.5 uCPU (*high-memory extra-large instances*). The initial configuration is generated randomly. To simulate a load spike, the uCPU demand is asked to grow for half the applications. To simulate transitions, 4% of the VMs have to be launched or resumed, 2% of the running VMs will be stopped or suspended, and 1% of the servers are being taken off-line. The estimated duration of each transition is: 1 second to launch a VM, 2 seconds to stop a VM, 4 to suspend, 5 to resume on the current server and 6 on another one. Finally, the migration of a VM lasts 1 second per gigabyte of RAM. For each instance, 10 minutes have been given to the plan module to compute one first solution on an Intel Xeon E5520 at 2.27 GHz running Linux 2.6.26-2-amd64 and Sun JVM 1.6u21 with 8 GB RAM allocated to the heap.

The tables hereafter display the average computational results by sets of 100 instances each: *solved* is the number of solved instances (failures are due to timeout), *obj* the average sum of the completion times in seconds, *nodes* the average number of nodes open in the search tree, *fails* the average number of fails, *time* the average solution time in seconds.

The consolidation ratio is the average number of VMs hosted per server. For this experiment, we simulated 5 ratio values by fixing the number of servers to 1,000 and varying the number of VMs from 2,000 to 6,000.

Table 1: Impact of the consolidation ratio on the solving process.

Ratio	Rebuild Mode					Repair Mode				
	solved	obj	nodes	fails	time	solved	obj	nodes	fails	time
`2:1`	100	452	2034	352	42.2	100	381	163	0	3.5
`3:1`	94	1264	3119	3645	75.2	100	749	394	0	8.4
`4:1`	65	3213	4574	11476	129.3	100	1349	836	0	18.7
`5:1`	10	7475	6878	47590	241.2	100	2312	1585	44	37.7
`6:1`	0	-	-	-	-	86	4092	2884	2863	71.5

Table 1 shows the impact of the consolidation ratio on the solving process in rebuild and repair modes. Increasing the consolidation ratio naturally makes the problem harder: the number of VMs to place rises up, making the packing tighter. The cost of the computed reconfiguration plan also grows as the migrations on the overloaded servers have to be more precisely orchestrated. The repair mode outperforms significantly the rebuild mode as it tackles, for a same ratio value, much smaller models. The results show that our policy for fixing VMs *a priori* in the repair mode is correctly balanced as it reduces well the model size without making the problem unsolvable, even for a consolidation ratio of `5:1`. Such a ratio implies an average CPU demand of 72% of the datacenter capacity. This

[7] http://aws.amazon.com/ec2

utilization rate is considered as ideal by system administrators as it provides an efficient tradeoff between a high resource usage and the ability to absorb the temporary load spikes.

The datacenter size. For this experiment, we generated 4 sets of instances using a fixed consolidation ratio of `5:1` and a variable datacenter size, from 500 servers and 2,500 VMs to 2,000 servers and 10,000 VMs.

Table 2: Impact of the datacenter size on the solving process (repair mode).

Set	#servers	#VMs	solved	obj	nodes	fails	time
`x1`	500	2,500	100	1160	805	13	7.0
`x2`	1,000	5,000	99	2321	1594	17	36.2
`x3`	1,500	7,500	99	3476	2374	43	105.5
`x4`	2,000	10,000	100	4635	3171	15	217.0

Table 2 shows the impact of the datacenter size on the computation in repair mode. We observe that the solving time grows non-linearly with the datacenter size, accordingly to the temporal complexity of the VRSP. The solver is however able to compute at least one solution for almost all the instances. Finally, the slow objective value growth and the few number of fails indicate the reliability of our search heuristics to guide the solver to solutions of high quality. These results show the ability of Entropy to handle large representative datacenter sizes. Indeed, the current trend in datacenter architecture consists in acquiring servers per shipping container. Each container is almost autonomous and contains between 500 and 2,500 servers[8]. While it is possible to aggregate several containers, *i.e.* several physical partitions, in one logical partition, the technical limitations of the platform software may prevent migrations between them. Entropy is then dimensioned to manage each partition individually.

The side constraints are now experimented in the context of Highly Available applications. For this experiment, one `spread` constraint is specified for each application tier to provide fault tolerance. One application asks for dedicated servers using a `lonely` constraint. Using `capacity` constraints, the hosted capacity of each rack is limited to 300 VMs. Maintenance are prepared using `ban` constraints on 0.5% of the running servers.

Table 3 shows the impact of the side constraints on the instances with a variable consolidation ratio (left) and with a variable datacenter size (right). For the highest consolidation ratio, the solver becomes unable to compute a solution. The packing is already tight and hard to solve, and the additional side constraints only exacerbate the situation. For lower ratios, the impact of the side constraints on the solving time and on the solution cost is quite acceptable. Until ratio `4:1`, the difference is not significant. With ratio `5:1`, the solver takes only 15 additional seconds to compute a solution subject to 750 `spread` constraints,

[8] http://www.datacentermap.com/blog/datacenter-container-55.html

Table 3: Impact of the side constraints on the solving process (repair mode).

variable consolidation ratios						variable datacenter sizes					
Set	solved	obj	nodes	fails	time	Set	solved	obj	nodes	fails	time
`2:1`	100	381	163	0	3.7	`x1`	97	1255	1156	3518	12.2
`3:1`	100	751	394	0	9	`x2`	93	2511	1872	3018	47.1
`4:1`	100	1376	841	31	19.2	`x3`	88	3778	2477	1670	120.2
`5:1`	95	2491	2007	7053	53.2	`x4`	91	4980	3271	957	238.7
`6:1`	35	4512	3603	9661	110.1						

20 `capacity` constraints, 5 `ban` constraints, and one `lonely` constraint, while the cost of the reconfiguration plan is increased by 13%. When the datacenter size varies, the impact of the side constraints appears again to be limited. With fixed ratio `5:1`, the solver is always able to compute a solution for more than 88% of the instances. For the largest problems, the solving time is only 9% greater than for the core VRSP, while the cost of the solutions is only 7% higher.

These experiments show that the impact of the side constraints is significant only when the core VRSP is itself already hard. In a well-designed datacenter, the primary bottleneck is the limited capacity of its servers. The side placement constraints should remain only tools provided to the administrators and clients to express their preferences. When they become preponderant, then the dimension of the datacenter should be rethought.

6 Related Works

Dynamic reconfiguration arises in real-time computing systems with two dominant applications: reallocation of storage devices to data and reallocation of processors to processes. In both cases, the goal is to improve the efficiency of the service as the system evolves, but the main concerns differ.

One key issue in *data migration* is when to schedule, given a final configuration and the limited capacity of the network, the transfer of data blocks to involve the least impact on the service. In the core problem, the reconfiguration time should be minimized, and each transfer occupies, simultaneously during one unit time, the unique ports of its sender and receiver devices. Such a *port constraint* is similar to the concurrent resource constraint of the RTSP. It is however simpler since it is a disjunctive resource constraint, uncorrelated with the storage capacity of the devices which is usually assumed to be unlimited during migration. In the Data Migration with Space Constraints [1] variant, both port and storage constraints have to be satisfied during the reconfiguration, but data blocks are assumed to be identical and the devices never full.

The key issue in *process migration* is rather where to redispatch the processes, given the limited capacities of the processors. Most former works, actually, only consider migrations by service disruption and thus are not subject to scheduling problems. CP-based approaches were proposed for two opposite objectives: Load Rebalancing [4] aims at finding a more balanced configuration while minimizing the number of migrations; Service Consolidation [3,6] aims at gathering the load in order to switch off the maximum number of unused processors.

Live process migration induces a scheduling problem with concurrent resource requirements during the reconfiguration. To our knowledge, this problem has previously only been studied in [6,8]. Sirdey et al. [8] presented the Process Move Program, a variant of the RTSP oriented to load balancing, with two transition types: unit-time live migration and migration by disruption. The resource requirements are constant and the final configuration is given. The problem is to minimize the number of disruptions and to order the live migrations of the remaining processes for solving the resource conflicts. The ZIPMP problem, evoked in Section 2, is the decision variant where no disruption is allowed. The authors provided a branch-and-bound and several metaheuristics solutions. In a previous implementation of Entropy oriented to consolidation [6], a fast schedule is searched in a greedy way, using a CP model to enforce the resource constraints to be satisfied at any time. The considered problem is an extension of the RTSP as it allows to migrate VMs on bypass servers to avoid cycles.

It turns out that, in former works, the packing and the scheduling parts of the reconfiguration problem have never been handled at once. Such a decomposition allows to deal with a quality criterion on the final configuration, namely load balancing or consolidation, but it hinders the objective to get fast reconfiguration plans. In the VRSP, consolidation and load balancing criteria could also be enforced as extra soft constraints or within the search heuristic.

The aforementioned reconfiguration problems match the exact resource constraints (1) of the BRSP, or a natural extension of them:

$$\sum_{\substack{j \in s_o^{-1}(r) \\ t < t_j}} b_j^o + \sum_{\substack{j \in s_o^{-1}(r) \\ t_j \leq t < t_j + d_{\delta(j)}}} b_{\delta(j)}^o + \sum_{\substack{j \in s_f^{-1}(r) \\ t_j \leq t < t_j + d_{\delta(j)}}} b_{\delta(j)}^f + \sum_{\substack{j \in s_f^{-1}(r) \\ t_j + d_{\delta(j)} \leq t}} b_j^f \leq B_r, \quad \forall r \in \mathcal{R}, \forall t \geq 0.$$

This extension allows the requirements to differ as the transitions are performed. Thus it allows to model disruptions ($b_{\delta(j)}^o = b_{\delta(j)}^f = 0$) as in Process Move Program or port constraints ($b_{\delta(j)}^o = b_{\delta(j)}^f = 1$ and $b_j^o = b_j^f = 0$) as in Data Migration. Furthermore our CP model can be extended to handle these constraints, using 4 task variables for each transition instead of 2. As a consequence, this model, with different objectives, fits most of the problems above described, at one notable exception: it cannot deal with bypass as in [1,6].

Regarding now the flexibility and the scalability of our approach, a dozen of relative placement constraints are currently implemented in Entropy. In the same context of datacenter resource management, Dhyani et al. [3] also advocated the power of CP technologies to handle some of these constraints. As previously said, they perform only consolidation, not reconfiguration, and experiment on instances with up to 30 servers with 4 resources each, and 250 VMs. Commercial VM managers propose also more and more services to their clients to express their needs in terms of placement. For example, the DRS [9] manager by VMWare performs consolidation and provides 3 affinity rules to customize the VM placement. These rules match the constraints, called in Entropy: `spread`, `ban` and its opposite, `fence`. A cluster managed by DRS can not exceed 32 nodes and 1280 VMs. The technology behind DRS is concealed.

7 Conclusion

Virtualized datacenters host and manage large ranges of applications, each application being distributed in VMs. The resource requirements of the VMs change over time. In addition, clients and system administrators have specific expectations regarding the relative placement of the VMs on the servers. Automatic reconfiguration is needed each time the current placement is no longer viable. Considering both the resource requirements and the placement constraints, the problem is to determine a new placement of the VMs and to schedule the transitions so as to provide a fast and reliable reconfiguration.

In this paper, we presented a general formalization of this problem, called Bin Repacking Scheduling Problem, and a model of Constraint Programming providing the flexibility needed to dynamically inject side placement constraints. Our model is implemented and integrated into the autonomous VM manager Entropy. Experiments with realistic simulated workloads show the ability of Entropy to solve problems involving up to 10,000 VMs on 2,000 servers with 900 side constraints in less than 5 minutes.

In future works, we want to enrich Entropy with more placement constraints, including constraints on the network topology, which could make our model drastically harder to solve. We aim also at providing side constraints with violation penalties, as clients prefer a controlled degradation of the service to any non-viable configurations. In addition, we want Entropy to be able to help a system administrator to locate issues, such as resource bottleneck CP provides, through soft constraints and explanations, the elements to address these needs. Their development will contribute to improve the usability of datacenters.

References

1. Anderson, E., Hall, J., Hartline, J., Hobbes, M., Karlin, A., Saia, J., Swaminathan, R., Wilkes, J.: Algorithms for data migration. Algorithmica 57(2), 349–380 (2010)
2. Beldiceanu, N., Carlsson, M., Rampon, J.: Global constraint catalog. Tech. Rep. 07, SICS (2010), `http://www.emn.fr/z-info/sdemasse/gccat/`
3. Dhyani, K., Gualandi, S., Cremonesi, P.: A constraint programming approach for the service consolidation problem. In: CPAIOR'10, LNCS, vol. 6140, pp. 97–101. Springer (2010)
4. Fukunaga, A.: Search spaces for min-perturbation repair. In: CP'09. pp. 383–390. Springer (2009)
5. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. WH Freeman & Co. New York, NY, USA (1979)
6. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a consolidation manager for clusters. In: VEE '09. pp. 41–50. ACM (2009)
7. Simonis, H., Cornelissens, T.: Modelling producer/consumer constraints. In: CP'95. LNCS, vol. 976, pp. 449–462. Springer (1995)
8. Sirdey, R., Carlier, J., Kerivin, H., Nace, D.: On a resource-constrained scheduling problem with application to distributed systems reconfiguration. European Journal of Operational Research 183(2), 546–563 (2007)
9. VMWare: Resource Management with VMWare DRS. Tech. rep. (2006)