Building Application Profiles to Allow a Better Usage of the Renewable Energies in Data Centres

Corentin Dupont

University of Trento Via Sommarive, 5 38123 Trento, Italy

Abstract. Data centres are powerful and power-hungry facilities which aim at hosting ICT services. The current trend is to, on the one hand, try to reduce the overall consumption of a data centre, and on the other hand to prioritize the utilization of renewable energies over brown energies. Renewable energies tend to be very variable in time (e.g. solar energy), and thus renewable energy aware algorithms tries to schedule the applications running in the data centres accordingly. However, one of the main problems is that most of the time very little information is known about the applications running in data centres. More specifically, we need to have more information about the current and planned workload of an application, and the tolerance of that application to have its workload rescheduled. In this paper, we present a work in progress on Plug4Green, a flexible VM manager able to reduce energy consumption in data centres. We extend Plug4Green with the second goal of increasing the usage of renewable energy in data centres. This includes the development of specific application profiles, and a new optimization technique.

1 Introduction

Data centres are large facilities which purpose is to host information processing and telecommunication services for scientific and/or business applications. Due to the rise in service demands together with energy costs, the energy efficiency has now been added as a new key metric for data centres. Energy-aware strategies are beginning to be integrated inside the data centre resource manager. In practice, a Virtual Machine (VM) placement algorithm considers the data centre and the workload characteristics to place the VMs among the servers in the most efficient way, considering performance and energy consumption. This placement must be done respecting the requirements of the Service Level Agreement (SLA) existing between the data centre and its clients.

In parallel to reducing the overall energy consumption, the current trend is to foster the use of renewable energies. Renewable energies have the problem to be very variable and time-dependent: for example solar power is available only during the day, and is subject to variations due to the meteorological conditions. Thus, data centre operators must try to shift the workload of running

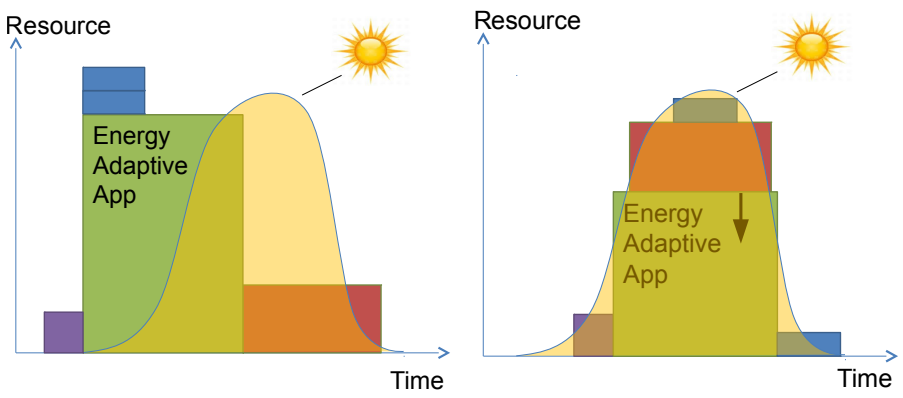

Fig. 1. Adapting applications for a better usage of renewable energies

applications in time, to match it with the availability (or forecast availability) of renewable energy, as it is depicted in Figure 1.

As preliminary work, we present Plug4Green [1], an energy aware VM manager based on Constraint Programming (CP) [2]. The use of CP allows to attain a relatively good flexibility and extensibility: indeed data centres are evolving permanently and new use cases are added regularly. We already proposed and implemented 23 VM placement constraints to address common concerns such as hardware compatibilities, performance, security issues, and workload instability. The usage of CP makes placement constraints, objectives, and algorithms independent from each other: new concerns can be added in the VM manager without changing the existing implementation.

The goal of this paper is to present a work in progress on the extension of Plug4Green, so as to extend its objectives to not only reduce the overall energy consumption of a data centre, but also allow a better usage of the renewable energies. A great challenge of efficiently using the renewable energies in a data centre is to be able to schedule correctly the workload of the applications. This shows the importance of being able to know the workload an application will have to run at a certain point of time, to understand under what conditions it can be shifted or delayed, and *in fine* to schedule it correctly.

Yet, currently most of the applications running in data centres are unaware of their self workload: they are unable to predict how much computing power they will require and when. In data centres, the knowledge of the requirements of an application in terms of resources is still "meta-knowledge", i.e. the knowledge of the data centre operators. For example, in data centres, database indexing maintenance operations are usually performed at night, to minimize the impact on the overall erformance. However, in a data centre using primarily solar power, it would be interesting to shift this task during the lunch break, when the sun is shining. The knowledge that this particular task, "database indexing", can cope with a 12 hour shift, and that it takes approximately half an hour, belongs to the operator's knowledge. In this paper, we show how this knowledge can be

encoded and used by Plug4Green to schedule the application workload correctly. We propose a design for the extended version of Plug4Green and discuss the possible optimization techniques.

The remainder of this paper is structured as follow: we will first perform a survey of the related works in Section 2. We then present the extended design of Plug4Green in Section 3 and preliminary implementation in Section 4. We conclude in Section 5.

2 Related Work

A few flexible and extensible frameworks for VM allocation have been proposed recently. For example, BtrPlace [3] is a CP-based flexible consolidation manager. Plug4Green leverages on Btrplace [4,5]. BtrPlace does not take into consideration energy related problems and does not provide an operator with the opportunity of setting optimization objectives. In contrast to BtrPlace, Plug4Green directly addresses energy consumption problem. This required numerous extensions: the development of a power model and different model extensions, two objectives with their associated heuristics, 7 energy-related constraints, and a domain-specific language to directly exhibit energy concerns and metrics such as PUE, CUE [1] and Watts, to the end-users.

Similar modular consolidation manager adopting CP paradigm is presented in [6]. The authors ensure high availability for VM placement by guaranteeing at any time a certain number of vacant servers to allocate VMs with regards to placement constraints. The manager scalability is effective for 32 servers and 128 VMs.

A hybrid system proposed in [7] solves a resource reallocation problem. This system includes Business Rules Management System (BRMS) and CP. A user can customize both business rules and constraints. The BRMS monitors and analyses the servers' state at a period of time to detect overloaded servers and bottlenecks. Once a problem is identified the BRMS models its instance and sends it to the CP solver which resolves it within seconds. In contrast to our manager, both the systems presented in [6] and [7] are not addressing energy-efficiency problems.

In [8], the authors proposes GreenSwitch, a model-based approach for dynamically scheduling the workload and selecting the source of energy to use. In this work, the authors focuses on the trade-offs involved in powering data centres with solar and/or wind energy, and propose an implementation of their solar powered mini data centre called Parasol. With contrast to this approach, we propose the possibility to schedule the workload at a finer grain, which is the application level.

[1] PUE and CUE are defined by The Green Grid Consortium: http://www.thegreengrid.org/

3 Design

We present the design selected for the Plug4Green prototype in Section 3.1. We then present our advancement in defining the application management engine in Section 3.2. We finally discuss the technology choice made for the optimization engine, and compare it especially to SMT, a technology that we envisage to use in the future development of Plug4Green in Section 3.3.

3.1 Plug4Green

Plug4Green is an extensible VM manager. The architecture chosen allows to easily extend the engine by adding new concerns, without modifying the underlying algorithms. In particular, new constraints can be added straightforwardly, as we showed by implementing 23 constraints commonly encountered in data centres, including energy-oriented ones. As can be seen in Figure 2, Plug4Green has the following inputs:

- The *SLAs*
- The *data centre configuration*
- A *Single Allocation* request
- Or a *Global Optimisation* request

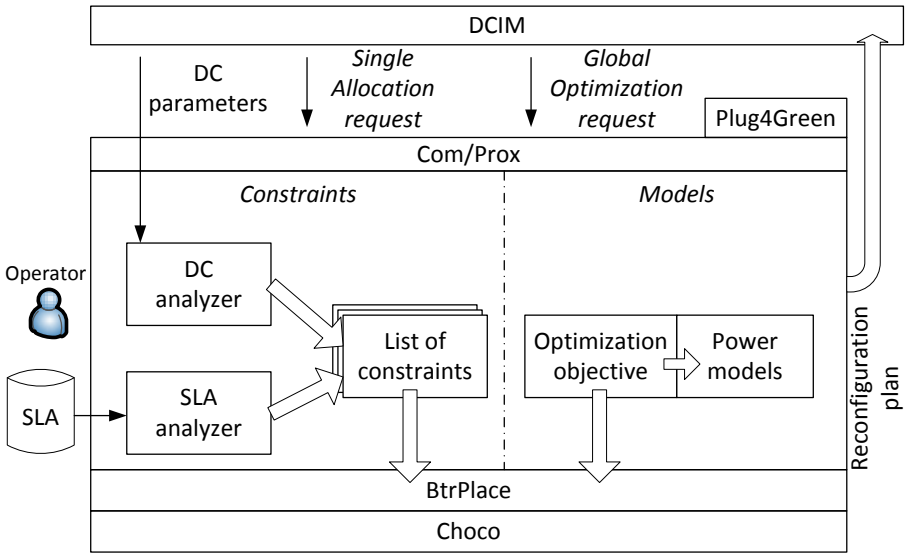

Fig. 2. Plug4Green architecture

Plug4Green considers a set of *SLA* constraints along with the *data centre configuration* to compute a *reconfiguration plan* as an output. The *data centre*

configuration captures all the relevant ICT resources of a data centre with their energy-related attributes and interconnections, in an XML format. The reconfiguration plan consists of a set of actions such as *powering on*, *powering off*, *waking up* and *putting in idle mode* a server, and *migrating* a VM, that satisfies all the constraints and minimizes the current objective. The objective can be to minimize either the power consumption of a federation of data centres, or the CO_2 emissions. The diagram shows the clear separation between the *Constraints* part ("what" we want to do) and the *Models* part ("how" to solve the problem), which is fundamental for extensibility.

Plug4Green is called by the Data Centre Infrastructure Management (DCIM) for two different events: *Single Allocation* or *Global Optimisation*. The *Single Allocation* event is triggered when a new VM have to be allocated. Plug4Green will compute and return the best server to allocate the VM on, taking into account the characteristics of the VM, the current state of the data centre, the SLAs and the current objective. The *Global Optimisation* event is itself triggered regularly (every ten minutes in our experimentation) and Plug4Green will return a reconfiguration plan. In manual mode, the data centre operator has the possibility to accept or reject this reconfiguration plan, while in automatic mode, it is enacted automatically. Plug4Green will then execute the reconfiguration plan in order to reduce the overall consumption of the data centre (either power consumption or gas emission) while also respecting the SLAs. The *Com/Prox* layer ensures that Plug4Green can be plugged easily to different existing DCIM: its the only part that must be updated when adapting the software for a new DCIM. Currently, Plug4Green can be integrated into VMWare[2], Eucalyptus[3], and HP Matrix Operating Environment[4] infrastructures. Plug4Green is based on the flexible consolidation manager BtrPlace [3].

We evaluated Plug4Green in an industrial test bed, to show that it is both efficient and scalable:

- Using our framework in a realistic cloud data centre environment allowed to reduce the overall energy consumption up to 33% and the gas emission up to 34%. These savings are achieved by considering the servers hardware heterogeneity, their different energy-efficiency and different compositions of SLAs.
- We showed by simulation how such an approach can be scalable. In particular, we were able to compute the improved placement of 7,500 VMs on 1,500 servers, while respecting their SLA.

3.2 Energy Aware Software Controller

In order to allow Plug4Green to optimize the usage of renewable energies, we extend the design of Plug4Green presented previously: we define the Energy

[2] `http://www.vmware.com`
[3] `http://eucalyptus.com`
[4] `http://h18004.www1.hp.com/products/solutions/insightdynamics/overview.html`

Aware Software Controller (EASC), as depicted in Figure 3. For each application, the EASC is in charge of:

- building an energetic profile of that application,
- defining the tasks and working modes,
- building the list of constraints,
- executing the activity plan as computed by Plug4Green.

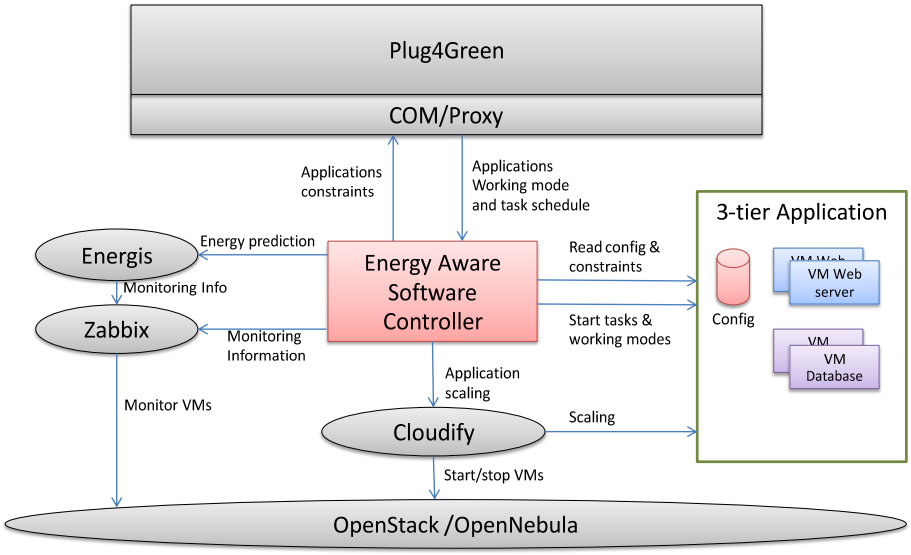

Fig. 3. Energy Aware Software Controllers for aPaaS

A working mode, specifically, is a particular way for an application to perform a task, according to its SLA. For example, a typical 3-tier application can have several VMs containing its web server, and be allowed to scale up or down the number of VMs according to the number of requests. Each possible combination of VMs is called a working mode. In order to build the energetic profile of the working modes, we use Zabbix to collect monitoring data for the VMs used by the application. We then use Energis[5] to compute the energy necessary for each working modes and tasks of the applications. Energis is a tool using predictive algorithms based on historical measurement data in order to predict the energy consumption of a particular VM.

The energetic profiles together with the defined working modes and tasks are then transmitted to Plug4Green, that will compute an optimized scheduling, called the activity plan. This activity plan is transmitted back to the EASC to be performed. In practice, the activity plan consists in spawning more or less VMs to execute the tasks of the application, such as front-end web servers or

[5] Energis: http://www.freemind-group.com/index.php/products/energis.html

back-end databases. A PaaS management tool such as Cloudify[6] can provide such a scalability service, together with OpenStack[7].

The EASC is instantiated into three flavours: The EASC aPaaS (Application Platform as a Service), showed in the picture, is in charge of controlling the Cloud applications inside a data centre. It will scale up and down 3-tier applications according to the availability of renewable energies. The EASC IaaS (Infrastructure as a Service) is in charge of collaborating with the Cloud management system to manage the data centre infrastructure. In practice, it we will tune the VM consolidation factor to allow more or less energy saving and thus follow the renewable energy availability. Finally, the EASC TM (Tasks Management) will shift in time the maintenance tasks that are performed by the data centre, such as virus scan or server decommissioning tasks. Those tasks will be scheduled when the renewable energy is available.

3.3 Optimization

Plug4Green is based on Constraint Programming, which is a programming paradigm devoted to solve Constraint Satisfaction Problems (CSP). In a CSP, relations between variables are stated in the form of constraints. Each constraint restricts the combination of values that a set of variables may take simultaneously. While CP was a very good choice and fulfilled most of the requirements, we discovered some practical drawbacks. Especially, defining new constraints easily is one of the main design goal of Plug4Green: as data centres evolves, new use cases arrives regularly, and a qualified operator should be able to insert new constraints into the engine. However, defining a new constraint takes a lot of lines of code and is also very error prone. The debugging period for each new constraint is also quite long. This diminishes the flexibility of the tool, which should imply the easy creation of new constraints to fit the new requirements arriving in a data centre.

To tackle this problem of flexibility, we started exploring alternatives to the couple Constraint Programming/Java. As an alternative to CP, we propose Satisfiability Modulo Theories [9] (SMT). A SMT problem is to determine the satisfiability of ground logical formulas with respect to background theories expressed in classical first-order logic with equality. Modern SMT solvers integrate a Boolean satisfiability (SAT) solver with specialized solvers for a set of literals belonging to each theory. The problem consists in finding an assignment to the variables that satisfy all constraints. We also surveyed the feasibility of using Pure Functional languages such as Haskell[8] as the base language for the constraint engine of Plug4Green. Programs in Haskell tend to be much less verbose than in Java (in the order of ten time less lines). It is also a declarative language, like Constraint Programming is, so the expression of constraints is more clear and natural. Furthermore, Haskell is pure, which combined with its strong type system allows to reduce drastically the number of bugs.

[6] Cloudify: http://getcloudify.org/
[7] OpenStack: https://www.openstack.org/
[8] http://haskell.org

4 Implementation

To show the usability of both SMT and pure functional languages to tackle energy efficiency problems in a flexible way, we implemented the classical problem of packing VMs on servers using the library SBV [9], with only one dimension for the sake of simplicity. In the example[10] showed in Listing 1.1, each VM has a demand in term of CPU, and each server has a certain CPU capacity to offer. The objective is to find the placement of the VMs on the servers that minimizes the number of servers needed. The only constraint applied is that the total CPU consumption of the VMs that will be running on a server must not exceed the capacity of that server.

```
1
2 --concrete IDs for VMs and servers
3 type VMID = Integer
4 type SID = Integer
5
6 --symbolic IDs of the servers
7 type SSID = SBV SID
8
9 --A VM is just a name and a cpuDemand
10 data VM = VM { vmName :: String,
11                cpuDemand :: Integer}
12
13 --a server has got a name and a certain amount of free CPU
14 data Server = Server { serverName :: String,
15                        cpuCapacity :: Integer}
16
17 --list of VMs
18 vms :: Map VMID VM
19 vms = fromList $ zip [0..] [VM "VM1" 100, VM "VM2" 50, VM "VM3" 15]
20
21 --list of servers
22 servers :: Map SID Server
23 servers = fromList $ zip [0..] [Server "Server1" 100, Server "Server2" 100, Server "Server3" 200]
24
25 --number of servers ON (which we'll try to minimize)
26 numberServersOn :: Map VMID SSID -> SInteger
27 numberServersOn = count . elems . M.map (./= 0) . vmCounts
28
29 --computes the number of VMs on each servers
30 vmCounts :: Map VMID SSID -> Map SID SInteger
31 vmCounts vmls = M.mapWithKey count servers where
32     count sid _ = sum [ite (mysid .== literal sid) 1 0 | mysid <- elems vmls]
33
34 --All the CPU constraints
35 cpuConstraints :: Map VMID SSID -> SBool
36 cpuConstraints vmls = bAnd $ elems $ M.mapWithKey criteria (serverCPUHeights vmls) where
37     criteria :: SID -> SInteger -> SBool
38     criteria sid height = (literal $ cpuCapacity $ fromJust $ M.lookup sid servers) .> height
39
40 --computes the CPU consummed by the VMs on each servers
41 serverCPUHeights :: Map VMID SSID -> Map SID SInteger
42 serverCPUHeights vmls = M.mapWithKey sumVMsHeights servers where
43     sumVMsHeights :: SID -> Server -> SInteger
44     sumVMsHeights sid _ = sum [ite (sid' .== literal sid) (literal $ cpuDemand $ fromJust $ M.lookup vmid
           vms) 0 | (vmid, sid') <- M.assocs vmls]
45
```

[9] http://leventerkok.github.io/sbv/

[10] The full implementation can be seen at https://github.com/cdupont/Plug4Green-design

```
46 --solves the VM placement problem
47 vmPlacementProblem :: IO (Maybe (Map VMID SID))
48 vmPlacementProblem = minimize' numberServersOn cpuConstraints
49
50 main = do
51   s <- vmPlacementProblem
52   putStrLn $ show s
```

Listing 1.1. Example of VM placement problem solved using SMT

When run, this program returns the placement for the VMs that minimizes the number of necessary servers. In this case, it will place all three VMs on the third server. While it is difficult to compare, it is anyway striking that this program is shorter than its equivalent in Java/Choco[11]. The definition of a constraint takes only a few lines (for example *numberServersOn* takes 2 lines) and flows with the program definition. Furthermore, as it is usually the case in Haskell, the type signature of the functions are carrying a lot of information that can be used both by the programmer to understand and reason about the program, and by the compiler to prove its correctness. For example, the type signature *numberServersOn :: Map VMID SSID -> SInteger* makes it clear that the function *numberServersOn* is a constraint that takes the positions of all the VMs on the servers (denoted as a mapping between the VM ids and the server symbolic ids) and returns a symbolic integer representing the necessary number of servers.

Furthermore, programming at the symbolic level, as it is required when designing a CSP, is not very different than programming in concrete Haskell. This is because a lot of the Haskell standard functions, like the function *sum* in our example program, can be reused in a constraint programming program. The definition of *sum* in the standard library of Haskell is generic enough to be able to be used also at the symbolic level. On the other hand, programming in Choco is completely different than programming in concrete Java: all the operators are necessarily different, due to the low genericity of Java. Therefore, the intuition of the Java programmer cannot be completely reused.

SBV is also a theorem prover, and that can be used to prove properties of the constraints expressed. For example, we might want to prove some properties about our constraint *vmCounts*. This function counts the number of VMs present on each servers. We want to prove the property that the count of VMs on a server has for absolute maximum the total numbers of VMs present in the data centre.

```
1
2 *Main> prove $ \x y -> bAll (.<= 2) $ vmCounts' [x, y]
3 Q.E.D.
```

Listing 1.2. Example of proof about a constraint

The listing 1.2 show how we can ask SBV to prove that the number of VMs per server computed by the constraint *vmCounts* cannot exceed the total

[11] for example this implementation of bin packing: http://www.dcs.gla.ac.uk/ pat/cpM/jchoco/binPack/CPBinPack.java

number of VMs (in this simplified example with only 2 VMs and a version of vmCounts defined for lists instead of maps). SBV simply replies with $Q.E.D$, showing that it found a proof of our property (this proof can be exhibited if needed).

In order to give to an optimization engine complementary informations about the profile of an application, we use a configuration file. An example is given in Listing 1.3 (written in Yaml).

```
1 Name: PetClinic
2 Tasks:
3   - Name: T1
4     Duration: 1h
5     StartTask: ./startT1
6     Dependencies:
7       - finishBefore T2
8       - finishBefore 00:00
9     WorkingModes:
10    - Name: W1D1
11      SwitchMode: ./switchMode.sh W1D1
12      Resources: m1.small
13      TimeFrame: WeekEnds
14    - Name: W1D2
15      SwitchMode: ./switchMode.sh W1D2
16      Resources: m1.small
17      TimeFrame: WeekDays, WeekEnds
18  - Name: T2
19    Duration: 2h
20    StartTask: ./startT2
21    Dependencies:
22      - finishBefore 00:00
23 Constraints:
24    - MutuallyExclusive: W1D1, W1D2
25    - AtLeastOne: W1D1, W1D2
```

Listing 1.3. Example of profile configuration file

The above listing describes a 3-Tier application, PetClinic, that has an Apache front-end, a Java back-end and a database. The front-end as well as the back-end can be scaled up and down using Cloudify: for example in the case too much web pages are requested, a new VM will be spawned and the Apache server will be installed in it using Chef[12]. The example file defines two tasks T1 and T2. Tasks define a punctual activity with a duration. We define a script able to start the task, and some absolute and relative dependencies. We also define working modes, W1D1 and W1D2, with the way we can switch from one mode to another (switchMode shell script), the needs in term of resources, and the possibly repetitive time frames during which those working modes are authorized. Finally we define overall constraints, such as "MutuallyExclusive", which describe a relation between two or more working modes that should not be active at the same time. Another example "AtLeastOne" describe the fact that there should be at least one working mode active at all time.

[12] http://www.getchef.com/

5 Conclusion

In this paper we presented our plan to enhance Plug4Green, an energy-aware VM manager based on Constraint Programming, in particular to allow it to increase the usage of renewable energies in data centres. We introduced the Energy Aware Software Controller, a new component communicating with Plug4Green and able to build energy profiles for Cloud applications, and to control the application following the activity plans computed by Plug4Green. To compute the activity plans, we surveyed the SAT Modulo Theory technique in order to integrate it inside Plug4Green.

Acknowledgments & Availability

The author would like to thanks the University of Trento, the EU FP7 projects FIT4Green and DC4Cities, and the Create-Net research centre. Evaluations presented in this paper were carried out by HP Innovation Centre Milan and INRIA.

Plug4Green is licensed under the terms of the Apache 2.0 License. The current prototype is available for download at `https://github.com/fit4green/Plug4Green`.

References

1. Corentin Dupont, Thomas Schulze, Giovanni Giuliani, Andrey Somov, and Fabien Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, pages 4:1–4:10. ACM, 2012.
2. Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.
3. Fabien Hermenier, Julia Lawall, and Gilles Muller. Btrplace: A flexible consolidation manager for highly available applications. *IEEE Transactions on Dependable and Secure Computing*, 10(5), 2013.
4. Fabien Hermenier, Sophie Demassey, and Xavier Lorca. Bin repacking scheduling in virtualized datacenters. In *Proceedings of the 17th international conference on Principles and practice of constraint programming*, CP'11, pages 27–41, Berlin, Heidelberg, 2011. Springer-Verlag.
5. Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50. ACM, 2009.
6. E. Bin, O. Biran, O. Boni, E. Hadad, E.K. Kolodner, Y. Moatti, and D.H. Lorenz. Guaranteeing high availability goals for virtual machine placement. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 700–709, 2011.

7. Roman Krogt, Jacob Feldman, James Little, and David Stynes. An integrated business rules and constraints approach to data centre capacity management. In David Cohen, editor, *Principles and Practice of Constraint Programming CP 2010*, volume 6308 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010.
8. Íñigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, and Ricardo Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. *SIGARCH Comput. Archit. News*, 41(1):51–64, March 2013.
9. Josep Suy Franch. Satisfiability modulo theories approach to constraint programming. 2013.