Towards Design-time Simulation Support for Energy-aware Cloud Application Development

Christophe Ponsard, Renaud De Landtsheer, Gustavo Ospina and Jean-Christophe Deprez
CETIC Research Centre, Avenue Jean Mermoz 28, 6041 Gosselies, Belgium

Keywords: Energy Efficiency, Cloud, Sustainability, Green-IT, Discrete Event Simulation, Self-adaptation.

Abstract: Cloud application deployment is becoming increasingly popular for the removal of upfront hardware costs, the pay-per-use cost model and their ability to scale. However, deploying software on the Cloud carries both opportunities and threats regarding energy efficiency. In order to help Cloud application developers learn and reason about the energy consumption of their application on the server-side, we have developed a framework centred on a UML profile for relating energy goals, requirements and associated KPI metrics to application design and deployment elements. Our previous work has focused on the use of such a framework to carry out our run-time experiments in order to select the best approach. In this paper, we explore the feasibility of a complementary approach for providing support at design time based on finer grained deployment models, the specification of Cloud and energy adaptation policies and the use of a discrete event simulator for reasoning on key performance indicators such as energy but also overall performance, delay and costs. The goal is to support the Cloud developer in pre-selecting the best trade-off that can be further tuned at run-time.

1 INTRODUCTION

The expansion of ICT both at professional and personal levels induces the processing and exchange of increasingly larger amounts of data, increasing connectivity of all devices (mobile devices, Internet of Things) and higher penetration in all domains. This would raise the energy required to run ICT to a dramatic level if ICT energy efficiency was not improving simultaneously. However, because of this continuous increasing of energy consumption (Internet Science NoE, 2013) and in order to reach another level of energy saving, it is required to consider the software layer. Several initiatives have already studied how to reduce energy consumption of mobile or embedded devices. For the Cloud Computing domain, an important amount of work has focused on lower layers such as the physical infrastructure (Dougherty et al., 2012) or on the infrastructure virtualisation layer (Mastelic et al., 2014). A systematic survey of sustainability showed a dedicated attention to Cloud as well as a number of proposals turned to the energy efficiency of software application (Penzenstadler et al., 2014). However much remains to be done, especially to help developers to learn how much energy is consumed by their application on the server-side.

In order to structure our work we will refer to the previously defined reference framework detailed in (Deprez et al., 2012). This framework is composed of three levels:

Requirements Level - We use the Goal-Question-Metric (GQM) paradigm (Basili et al., 1994). In previous work (Deprez and Ponsard, 2014), we have shown how developers can formulate energy-related goals and questions in order to gain a more precise knowledge of the energy consumed by various features/components of their application. We also make the link with a number of already identified energy-related metrics (Bozzelli et al., 2013).

Design Level - To capture the information in a way that is both standard for the analyst and easy to process in further steps, a UML profile was defined (Ponsard et al., 2015). This profile enhances the analysis process of a Cloud application with energy awareness both for the development of a new application or the migration of an existing application to the Cloud. It also enables the automated deployment of measurement probes to monitor the specified Key Performance Indicators (KPI) and report them at the GQM level.

Run-time Level - Probes collect the specified data and report them to a monitoring infrastructure part of the energy-aware Cloud stack. For this purpose we currently use the ASCETiC Cloud stack deployed in specific test beds (ASCETIC, 2013).

So far, our work has mostly been oriented towards

Ponsard, C., Landtsheer, R., Ospina, G. and Deprez, J-C.
Towards Design-time Simulation Support for Energy-aware Cloud Application Development.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 2*, pages 398-404
ISBN: 978-989-758-182-3

run-time approach to collect data about the energy behavior as well as other key non-functional requirements such as performance and response time that require some form of trade-off. While this approach is the most precise given it collects real data, it suffers from some drawbacks. First, it is necessary to reach the deployment phase to collect data: although Cloud architectures are quite flexible, this arrives quite late in the process and it reduces the set of alternatives for deployment configuration and might rule out more fundamental design alternatives. Second, it is difficult to experiment with self-adaptation policies, especially when considering the most abstract SaaS level. Third, collecting data might also be difficult and be subject to some measurement bias. Given that our approach already proposed a strong design-level modelling, we felt that it might be interesting to propose complementary tools supporting the assessment of architectural alternatives directly at the design level. Our approach does not aim at being very precise; it only aims at reducing the design space for the SaaS level developer. At the technical level, the idea is to rely on simple Cloud simulations not requiring a full blown Cloud simulation like CloudSim (Calheiros et al., 2011) or GreenCloud (Kliazovich et al., 2010).

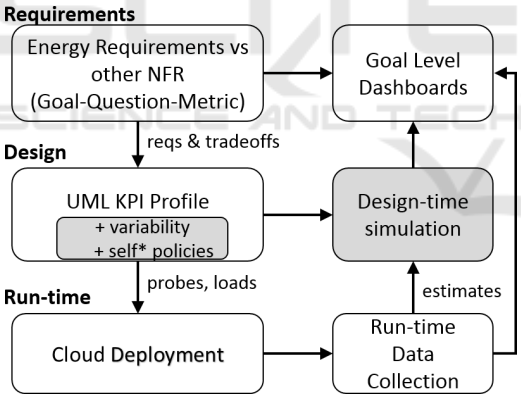

Figure 1: Extended design-time approach.

Our extended approach is highlighted in Figure 1, where the contribution discussed in this paper is highlighted in gray. It includes a new simulation tool and relies on an extended UML model. This extension mainly consists of refinement of the existing deployment view with:

- more information enabling the estimations of different kinds of requirements (throughput, delay, energy per requests). Those can be produced by the run-time layer through unit tests.
- self-adaptation policies, in order to determine the dynamic behavior, e.g. to scale but also to preserve energy.
- variability points, enabling to compare different alternatives. Such variability points can be structural ones or related to adaptation policies.

The purpose of this paper is mainly to assess the feasibility of the approach which relies on three technological locks. First it is necessary to map the deployment model into a realistic simulation model. Second it is necessary to define queries on this model and evaluate them efficiently. Third the results should be precise enough, compared to run-time simulation, to be useful.

This paper aims at partially solving these three technological locks. Section 2 details the finer grained simulation model and engine on which our UML deployment model is mapped. Section 3 describes the query language that is used to efficiently express the targeted non-functional requirements. Section 4 details the first experiments we conducted so far using an Open Source discreet event simulator (DES) and a Photo Album web-application. Section 5 discusses some related work. Finally, section 6 draws some conclusions about our experience so far and identifies further work.

2 FROM UML TO SIMULATION MODEL

The main elements of Cloud application are represented in a simulation meta-model which allows us to define concrete models that are simulated in a Discrete Event Simulation engine. This model is close to a UML deployment diagram and maps easily with it. It introduces some refinement needed for the simulation such as the need to define interfaces between processes, either as buffers of storage devices. Those can be made transparent in the mapping process but are still explicit in our current version of the engine. We detail two main elements: structural parts and dynamic parts (policies driving specific behaviors such as scaling up and down).

2.1 Structural Modelling

In our approach, a Cloud application is modeled as flows of requests through processes and buffers/data stores. We do not specialise further our description although there are well-known classes of architectural processes in Cloud application such as load balancers, authentication gateways, workload schedulers, etc. We let those be organised at a higher ”pattern” layer.

Batch Processes are data processing services working as much as they can as they are fed by input

data (or requests) to produce some output results. Input data is received from one or different sources (and possibly requiring to wait for a specific resource), processed for some time that can be defined using a stochastic function, and finally the produced outputs are dispatched to their respective destinations, either an input buffer of another process or a data store.

Continuous Processes are pipelined processes: requests are continuously picked from input data stores and pass through a number of steps before being output.

Splitting Processes are similar to batch processes, except that they have several sets of outputs and when it completes, one set of output is selected and the produced items are dispatched to associated output. A splitting process can also represent a batch process that has a failure rate.

Parallel Processes are a variant of the above processes where several lines of the same process are running in parallel (either as threads or in different machines but with a dedicated processing resource). Specific adaptation policies can also be associated to dynamically change the level of parallelism of a process.

Buffers and Stores represent any kind of memory device able to store data. They have a maximum capacity. When this capacity is reached, they either overflow and lose data (e.g. rotating buffer), or block the upfront processes until it is able to cope with the request. Processes should always be isolated either by a store (indirect coupling) or a buffer (direct coupling).

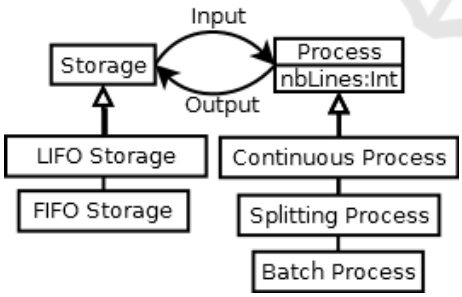

Figure 2: Concepts of our process modelling languages.

Data items flowing in processes and stores are indistinguishable at a given point of the processing chain. Yet, they have some intrinsic features: some items might come from a given process, others might have higher priority, etc. These intrinsic features can influence on the behavior of some processes, such as processing higher priority items first. This notion of intrinsic features also leads to the natural distinction between two different types of stores, namely: First In-First Out (FIFO) stores and Last In-First Out (LIFO) stores.

2.2 Process Activation and Self-adaptation Policies

Policies are also integrated in our model, together with activation policies that are able to turn a process on or off. To model these two concepts, we introduce the notion of *activable* and *activation*. An activable is something that can be enabled through an activation. We also associate a magnitude with the activation, that is, an integer. An activable can be a process or an external request. In the case of a process, the activation represent the number of batches that the process is allowed to execute. In the case of an order, the magnitude represent the number of requests.

Activable entities can be activated based on three types of rules that are also part of our modelling framework:

- regular activations, that perform the activation on a regular basis, based on a period of time;
- request-based activations, that perform the activation when a request is received;
- monitoring activations, that perform the activation when some conditions are met, possibly in a future time.

While the first and second class are useful to simulate loads, the later is the most interesting to specify the dynamic behaviour of the Cloud application as described later.

3 TRANSLATING ENERGY RELATED REQUIREMENTS INTO MODEL QUERIES

The goal of our approach is to perform queries related to non-functional requirements (NFR) on factory simulations, especially to measure energy and specific trade-offs e.g. related to performance. These queries are meant to be performed on single runs of simulation occurring inside the Monte-Carlo engine which aggregates the queries results over the runs, which can then be queried afterwards to obtain statistics like the mean, median, extremes, and variance.

Our query language can roughly be split into six sets of primitives, namely: probes on processes, probes on stores, logic operators, temporal logic operators, arithmetic operators, and temporal arithmetic operators. Arithmetic and logic operators differ by their return types; they return numeric and boolean values, respectively. Since this query language runs over simulated time, we take the convention that the value of the queries are computed at the end of the

trace on which they are evaluated. The full semantics of our query languages has been reported for an application in the supply chain domain in (De Landtsheer et al., 2016). It relies on the $\models$ notation: $t \models P$ is the value of expression P when evaluated at position t of the current trace. In this section, we highlight the adaptation of this language to the Cloud model elements presented in the previous section.

3.1 Probes for Processes

The probes on processes are atomic operators that extract basic metrics from processes of the simulation model. Suppose that p is such a process, the following probes are supported:

- $t \models \text{total}(p)$ the total number of instances in the process pool at time t.
- $t \models \text{running}(p)$ the number of running instances in the process pool at time t.
- $t \models \text{processedRequests}(p)$ the total number of requests processed by the process between the beginning of the trace, and time t.
- $t \models \text{incommingRequests}(p)$ the number of incoming requests to the process between the beginning of the trace, and time t. For a process with multiple lines, it sums up the started batches of each line.
- $t \models \text{totalWaitDuration}(p)$ the total duration where the process was not running between the start of the trace, and time t. for a process with multiple lines, it sums up the waiting time of each line.

3.2 Probes for Buffers and Stores

The probes on stores are atomic operators that extract basic metrics from stores of the simulation model. Suppose that s is such a store:

- $t \models \text{empty}(s)$ true if the store s is empty at time t, false otherwise.
- $t \models \text{content}(s)$ the number of items in the store s at time t.
- $t \models \text{capacity}(s)$ the maximal capacity of s. This is invariant in time.
- $t \models \text{relativeCapacity}(s)$ the relative content of store s at time t, that is: the content of the stock divided by the capacity of the store.
- $t \models \text{totalPut}(s)$ the number of items that have been put into s between the beginning of the simulation and time t, not counting the initial ones.
- $t \models \text{totalFetch}(s)$ the number of items that have been fetched from s between the beginning of the simulation and time t.
- $t \models \text{totalLostByOverflow}(s)$ the number of items that have been lost by overflow from s between the beginning of the trace, and time t. If s is a blocking store, this number will always be zero.

3.3 Operators

Complex queries can be built using queries in Object Constraint Language (OCL) enriched with the following operators, some of them also referring to one or more states of the considered trace:

- logical: $\text{true}, \text{false}, \mathit{not}(!), \mathit{and}(\&), \mathit{or}(||), <, >, \ldots$
- temporal logic: $\text{hasAlwaysBeen}, \text{hasBeen}, \text{since}, \ldots$
- arithmetic: $+, -, *, /, \mathit{sum}, \ldots$
- temporal arithmetic: $\text{delta}, \text{cumulatedDuration}, \text{time}, \text{min}, \text{max}, \text{avg}, \text{integral}\ldots$

4 FIRST EXPERIMENTS ON A CASE STUDY

Our first validation experiments have consisted in checking the ability to model our photo album case study in our simulation framework. This includes: using the given structural primitives, precisely describing queries related to specific requirements (especially energy and performance related), modelling some dynamic policies, and running a simulation based on an Open Source simulation engine.

4.1 Case Study Description

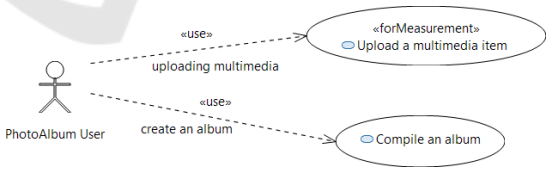

Figure 3: Extended design-time approach.

Photo Album is a 3-tier web application that is designed to be desktop-like on-line photo manager (Tsebro et al., 2009). It provides social services for uploading photos, storing and previewing them, creating albums and sharing them with other users. The visualisation layer is implemented in JavaScript while the business logic in Java runs on the server-side and a database for storing issue data can run on the same server or on a different machine. It is very representative of applications that can be deployed on a Cloud by SaaS models that can benefit of the PaaS and IaaS layers elasticity/reconfigurability features. Figure 3

present two typical (and related) use cases on this system: uploading a media item and compiling an album.

4.2 Structural Modelling

In order to model the Photo Album, we introduced the following model elements which are also depicted in Figure 4:

- an external user pool simulating an external load on the system with some stochastic behavior
- a *FRONT* process responsible of authentication and redirecting the request to the relevant service with a non-blocking buffer in front of it to cope with peak requests.
- an *UPLOAD* process simulates the media upload. It features high delay (between 1 and 2 min to complete to upload).
- a *COMPILE* process to compile an album which is triggered on demand or when a new media upload as occurred.
- an overflowing store is used to simulate an infinite storage space available for the two previous processed (no matter whether it overflows, this fact can be discarded in the analysis). The energy cost to write in the store is related to the size attribute of the data transferred to it.

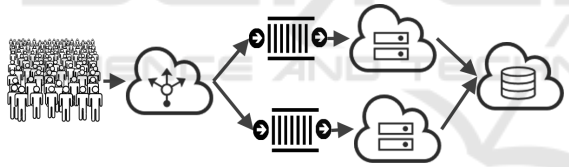

Figure 4: Photo album model.

4.3 Implementation

The described model was implemented using the Discrete Event Simulation (DES) engine of the Open Source OscaR library (OscaR, 2012). OscaR is a multi-purpose library completely written in Scala and composed of several complementary tools for operational research. The OscaR.DES engine is very efficient at computing the evolution of a system by evaluating changes only when they occur and updating complex expressions on the system. The engine can also be encapsulated into a web server and also features a graphical web-client initially developed for simulating supply chains (De Landtsheer et al., 2016). Note that the server time granularity is arbitrary and that simulators can dynamically adapt to the lowest granularity. The overall performance is related to the frequency of fined grained events.

4.4 Expressiveness

In order to assess the expressiveness we considered the capture of the following specific energy, performance and responsiveness requirements:

- total energy (on whole system):

```
sum(Process.allInstances()
  ->select(p:Process|p.energy*p.processedRequests))
```

- missed request ratio (on the *FRONT* process):

```
FRONT.processedRequests/FRONT.incommingRequests
```

- average response time (on a chain):

```
avg(sum(ProcessChain.allInstances()
   ->select(p:Process | p.runtime)))
```

We also assessed how self-adaptation policies could be captures in the model. Such policies are captured using activation policies associated with specific model elements. Policies are expressed in the form $WHEN\ condition\ THEN\ action$. The *condition* part is a query on the model while the *action* part is some modification on the model instances such as changing some parameter ($Process.add/removeInstance$). We considered the following policies:

- Scale up the pool of *UPLOAD* processed when its buffers are filling up:

```
WHEN MEDIA_buf.capacity\>0.8*MEDIA_buf.MAX
THEN MEDIA.addInstance(1)
```

- Scale down for energy efficiency:

```
WHEN HasAlwaysBeen(MEDIA.running\<0.8*MEDIA.total,10min)
$THEN MEDIA.removeInstance(1)
```

- Enforce maximal update frequency of album compilation:

```
WHEN HasBeen(changed(STORE.data(album)),3min)
THEN AVOID COMPILE(album)
```

Most of the policies can be supported out of the box by the DES engine except for negative activation policies (AVOID keyword in the last policy). We feel it is important to be able to capture such policies for enforcing energy behavior restrictions.

4.5 Simulation Run

Some simulations were executed based on the model described before. At this stage we could successfully encode the previously described probes. A number of default probes can also automatically generated from specific templates (including energy related). Figure 5 shows the results for an energy probe over 100 simulations, gathered using Monte Carlo with some variability in the delays between incoming requests. The

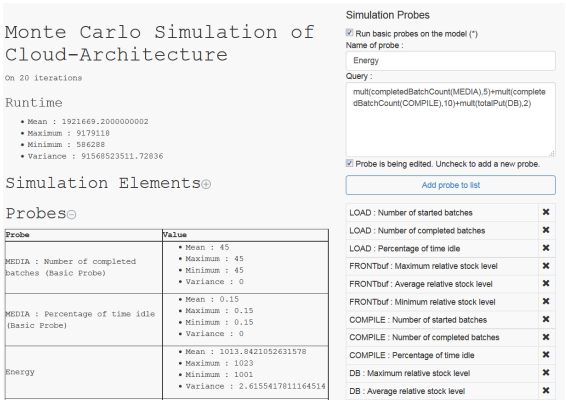

Figure 5: Monte Carlo simulation run with energy probe.

simulation takes a few seconds with about 10 percent overhead for the set of about 20 measured probes.

At this point, we are able to simulate and compare different design alternatives, either based on distinct models or using parameters available as variability point (for example, a load balancing component can have some threshold to trigger new client or to perform request aggregation). However at this point, the exploration of the design space is not yet automated: each alternative needs to be simulated manually. For example, Table 1 shows the impact of a buffer size to optimize energy in a context similar to Figure 4.

Table 1: Exploration of buffer size parameter to minimize energy consumption.

Buffer size	CPU Load	Network Traffic	Energy Cost
0	30%	1000	67
10	64%	100	19
20	95%	50	11.6
30	99%	33	11.1
40	100%	25	11

CPU load should be maximized as it is more efficient at high load and network traffic should be minimized. The simulation results confimrs this and shows the buffer size can safely be set around 30 requests to minimize energy consumption. However, beyond that limit, there is no gain because the bottleneck becomes the process itself.

5 RELATED WORK

There are a number of Cloud simulators available on the market and some of them are more specifically dedicated to energy efficiency. CloudSim (Calheiros et al., 2011) is one of the most popular cloud simulators. It operated at IaaS level and supports the simulation of virtualised data centres mostly focusing on computational intensive tasks, data interchanges between data centres. GreenCloud (Kliazovich et al., 2010) and iCanCloud (Núñez et al., 2012) are two other popular IaaS simulators based on network simulators, allowing greater precision at this level but are also more heavyweight. GreenCloud supports precise energy estimation of computation and network operations. Overall, all those simulator are hard to employ and often suffer from performance issues. DISSECT-CF (Kecskemeti, 2015) is a more recent simulator build to support easy extensibility, energy evaluation of IaaS and fast evaluation of many simulation scenarios. However, compared with our approach, all those simulators are limited to the IaaS layers and do not allow to reason on higher level abstractions defined at PaaS or SaaS layers, e.g. the use of Cloud Computing patterns (Fehling et al., 2014) and to experiment with self-adaptation strategies at that level. Our approach sacrifices the precision in order to gain such capabilities.

There are some SaaS level simulators closer to our work. AppSim (app, 2012) is an application simulator that models multi-tenant SaaS applications features. However it does only cope with that scenario and only focus on performance and not yet on energy efficiency. EffSim (Prabhakar et al., 2011) features a DES approach to tackle energy efficiency aspects but considers the more specific problem of large scale storage systems. On a more practical side, NICTA has developed a technology for modeling the performance and scalability of Cloud applications (**?**). In-depth empirical evaluations were conducted on a variety of real cloud infrastructures, including Google App Engine, Amazon EC2, and Microsoft Azure to predict the resource requirements in terms of cost, application performance, and limitations of a realistic application for alternative deployment scenarios.

6 CONCLUSIONS AND FUTURE WORK

In this short paper, we proposed an extension to our framework to better support Cloud developers in producing energy-aware applications. Our extension aims at providing more design time support based on the already present modelling artefacts and enabling early simulation of possible design alternatives directly at the SaaS level of abstraction. The drawback is to accept sacrifice in precision. Thus the proposed approach does not aim at replacing more precise simulations or run-time experiments, but we feel it could complement such approaches nicely by helping the Cloud architect to focus on the right design choices.

Our preliminary experiments carried out so far are encouraging as they can capture interesting interacting requirement and model adaptation policies and thus support the exploration of design trade-offs. The simulations are also very reactive but were not yet applied to very large models. Our next step is to validate how precise are the results and how key parameters required for the simulation can be estimated, especially in cooperating with our run-time approach. The DES model also needs to be elaborated: it is still partial and not fully aligned with the Cloud domain: more specific attributes needs to be identified and captured, e.g. for also reasoning on some security impacts. The model mapping also needs to be automated either through a domain specific graphical editor or the transformation of the existing UML deployment model, enriched with specific annotations.

ACKNOWLEDGEMENTS

This work was partly funded by the European Commission under the FP7 ASCETiC (nr 610874) and the CORNET SimQRi (nr 1318172) projects.

REFERENCES

(2012). AppSIM: An Application Simulator for Software as a Service (SaaS). In *Int. Conf. on Computational Techniques and Mobile Computing*.

ASCETIC (2013). Adapting Service lifeCycle towards EfficienT Clouds. http://www.ascetic.eu.

Basili, V. R., Caldiera, G., and Rombach, D. H. (1994). *The Goal Question Metric Approach*, volume I. John Wiley & Sons.

Bozzelli, P., Gu, Q., and Lago, P. (2013). A systematic literature review on green software metrics. Technical report, Technical Report: VU University Amsterdam.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya, R. (2011). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50.

Deprez, J.-C. and Ponsard, C. (2014). Energy related Goals and Questions for Cloud Services. In *Measurement and Metrics for Green and Sustainable Software (MeGSuS)*.

Deprez, J.-C., Ramdoyal, R., and Ponsard, C. (2012). Integrating Energy and Eco-Aware Requirements Engineering in the Development of Services-Based Applications on Virtual Clouds. In *First Int. Workshop on Requirements Engineering for Sustainable Systems*.

Dougherty, B., White, J., and Schmidt, D. C. (2012). Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Comp. Syst.*, 28(2):371–378.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014). *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Springer.

Internet Science NoE (2013). D8.1. Overview of ICT energy consumption. http://www. internet-science.eu

Kecskemeti, G. (2015). Dissect-cf: A simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58, Part 2:188 – 218. Special issue on Cloud Simulation.

Kliazovich, D., Bouvry, P., and Khan, S. (2010). Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, pages 1–21.

De Landtsheer, R., Ospina, G., Massonet, P., Ponsard, C., Printz, S., Hartel, L., and von Cube, J. P. (2016). A Discrete Event Simulation Approach for Quantifying Risks in Manufacturing Processes. In *International Conference on Operations Research and Enterprise Systems (ICORES16)*.

Mastelic, T. et al. (2014). Cloud computing: Survey on energy efficiency. *ACM Comput. Surv.*, 47(2):33:1–33:36.

Núñez, A. et al. (2012). iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator. *J. Grid Comput.*, 10(1):185–209.

OscaR (2012). OscaR: Scala in OR. https://bitbucket.org/oscarlib/oscar.

Penzenstadler, B. et al. (2014). Systematic mapping study on software engineering for sustainability (SE4S). In *18th Int. Conf. on Evaluation and Assessment in Software Engineering, EASE '14, London, UK, May 2014*, pages 14:1–14:14.

Ponsard, C., Deprez, J.-C., and Flamand, J. (2015). A UML KPI Profile for Energy Aware Design and Monitoring of Cloud Services. In *10th Int. Conf. on Software Engineering and Applications (ICSOFT-EA)*.

Prabhakar, R., Kruus, E., Lu, G., and Ungureanu, C. (2011). Eeffsim: A discrete event simulator for energy efficiency in large-scale storage systems. In *Energy Aware Computing (ICEAC), 2011 International Conference on*, pages 1–6.

Tsebro, A., Mukhina, S., Galkin, G., and Sorokin, M. (2009). Rich faces photo album application. http://docs.jboss.org/richfaces/latest_3_3_X/en/realworld/html_single.