Portable Green Cloud Services

Stephan Ulbricht[1], Wolfram Amme[1], Thomas Heinze[1], Simon Moser[2] and Hans-Dieter Wehle[2]
[1]*Friedrich-Schiller-University Jena, Jena, Germany*
[2]*IBM Germany Research & Development GmbH, Böblingen, Germany*

Keywords: Cloud Computing, TOSCA, Green IT, Policy, Energy Efficiency, Standardized Cloud Services.

Abstract: The areas of cloud computing and green IT are amongst the fastest growing markets in the IT industry, until now there are very few opportunities to combine the potential of both. In this paper, we present a method to combine the advantages of both to create standardized and energy efficient cloud services. For their description, we will use the cloud computing standard TOSCA(OASIS, 2013). Thereby, it is possible to create standardized and model-based cloud applications which can be deployed in many cloud environments. We will further show how it is feasible to combine policies with TOSCA to realize energy-efficient management of cloud services. To accomplish this, we will provide ideas on how to extend the TOSCA language as well as the cloud operating environment to achieve the goal of portable, energy-efficient cloud services. The core of this work is the identification and integration of the underlying system architecture for a common solution concept. For this, the architectures and necessary adjustments are explained.

1 INTRODUCTION

In times of climate change and rising energy costs, even IT companies (who are traditionally not in the spotlight when it comes to naming the biggest polluters) start to think about improving their energy efficiency. While moving a traditional IT application from an data center to a modern, energy-efficient virtualized cloud implementation already has a positive impact on the energy bill, such a step also allows to improve the automated energy-management of the application. In practice, there are two concerns: a.) for the consumer, moving an application to the cloud practically means committing to one cloud provider, as there is no standardized description format that helps to avoid a provider lock-in, and b.) in terms of energy-efficiency, there is no way for a cloud service provider to formally describe the energy guidelines that he would like to have followed. So neither the consumer nor the provider of a cloud service have the possibility to influence the environmental friendliness of an IT service - they are dependent on what the data center or cloud provider offers, both in terms of the infrastructure and the energy sources that get used as well as during the actual operation of the service.

This a huge opportunity to give both cloud service-consumers and -providers a choice: For a cloud service consumer to state that he wants to use ”green” cloud services, and for cloud service providers to model the service with a ”green” mindset as well as to influence or define the operation of an actual service instance. The first thing that is needed to address this challenge is an accurate model of the service. With model-based cloud services, the dependency on a specific data center or cloud provider can be reduced (since they are portable), and if these cloud services contain energy guidelines, such guidelines could even be reflected when the service is operated. Chapter 3 introduces TOSCA(OASIS, 2013), a standard for describing such model-based cloud services.

2 RELATED WORK

The efforts regarding energy-efficient and environmental sustainable IT solutions have increased significantly in recent years and many different approaches and techniques have been developed. There are three categories where energy improvement actions can be undertaken: Hardware, Software and IT environment.

On the *hardware* side the simplest method for a reduction of energy consumption is the shutdown of unused devices. Such techniques are shown in (Hwang and Wu, 2000, Sueur and Heiser, 2010, Dargie, 2012). In the *software* category it is possible for multiple users to work simultaneously on the same hardware,

Ulbricht S., Amme W., Heinze T., Moser S. and Wehle H..
Portable Green Cloud Services.
DOI: 10.5220/0004787300530059
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 53-59
ISBN: 978-989-758-019-2
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

through virtualization of computing resources. Research showed different approaches how environmental sustainability can be achieved by process scheduling - c.f. (Duy et al., 2010, Wang et al., 2010). The *IT-Environment* includes a number of components and characteristics, ranging from the power supply via the used cooling system to the layout of the components. Various papers have dealt with the subject of reducing fossil energy sources (Zhang et al., 2011, Liu et al., 2011). These approaches propose to allocate IT requests depending on available energy sources (wind, solar, etc.) or the overall energy costs.

To achieve a sustainable energy improvement of data centers, a holistic solution spanning hardware, software and IT-environment is required. Analogous to our solution, there is other research in this field, presenting frameworks and architectures to realize green enhancements in cloud computing environments - c.f. (Younge et al., 2010, Buyya et al., 2010, Beloglazov et al., 2012, Berl et al., 2010). To create a powerful and widely used solution concept, it is necessary to provide a standardized approach. This is the prerequisite for the acceptance and dissemination in the economy. In the remainder of this paper we'll show how this can be achieved with the help of cloud standards and policy languages.

3 CLOUD STANDARD: TOSCA

Chapter 1 already outlined the need for standardized, model-based cloud services. To ensure the interoperability and portability of cloud services, the standards body OASIS has started the standardization initiative TOSCA (Topology and Orchestration Specification for Cloud Applications) (OASIS, 2013). TOSCA is a meta-language for the model-based description of cloud services. This language not only allows the definition of the topology of an application, but also of its management behavior by requiring business processes which are operating directly on the application topology and thus define the operational semantics. The aim of TOSCA is to create standardized cloud services and simplify exchangeability. These standardized cloud services then get executed by a runtime entity called a "TOSCA container", which e.g. interacts with a hypervisor to install the topology and interprets the management plans to manage an actual service instance. As Figure 1 shows, a TOSCA-based cloud service is described by a *service template*, which is made up of a *topology template* and a (set of) *plan(s)*. The *topology template* defines the structure of the service, i.e., the base components and their relationships. A component can be a VM, an OS, a network connection, etc.

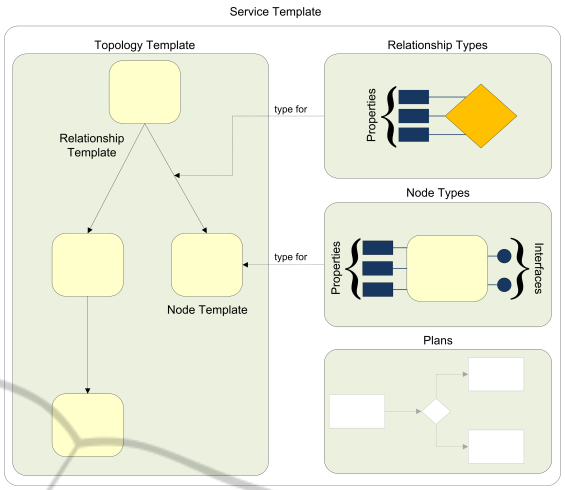

Figure 1: TOSCA service template.

Build- and Management Plans define the business process models which control the operational management behavior of the service, operating on the topology. If modeled correctly, the operational management behavior spans the complete life cycle of a service: A build plan for setting up the service, plans for any operation during its lifetime (e.g., trigger backups, install patches, elastic scaling, ...), as well as a termination plan when the service gets deprovisioned. The scope and the implementation of the plans are specific to the service and determined by the service modeler. When finished, the service modeler creates a *Cloud Service Archive* (CSAR), a package that contains the topology template, the plans and all other required artifacts (e.g., image files, scripts, etc.). These CSAR files can then be exchanged between different IT environments. More precisely, a topology template can be represented by a graph, where the nodes are the base components (e.g. a VM) and the edges are the relationships between the components. Both nodes and relationships are first defined outside of a concrete service, as they usually represent reusable building blocks. They are called *Node Types* and *Relationship Types*. A *Node Type* is a node of the type "VM" when used outside of a topology, but once added to a template (potentially with some properties attached), it becomes a *node template*. The same is true for the edges of the graph: *Relationship Types* define abstract relationships (like "hosted on" or "connected to"), but when used in a topology template they become *relationship templates*, describing the relationship between the node templates and also defining the potential states of a service instance.

The structure and functionality of service templates will be shown by example of an LAMP-infrastructure. The corresponding topology template

has the elements ”Linux OS”, ”Apache Web Server”, ”MySQL Database” and ”PHP Module”. The web server and the database are connected via the relationship ”hosted on” with the operating system, and so is the PHP module with the web server. The underlying relationship template ”hosted on type” is characterized by the source and destination element. The second type of relationship (connects to) exists between the PHP module and the database. The description of a node type, e.g. the Web Server, defines properties such as the IP address of an instance.

Another feature of TOSCA is the fact that it allows policies for certain elements, e.g. for *Service Templates* and *Node Templates*. Policies enable the dynamic control of the behavior, i.e. rules can be defined for access control (authorization policy) or to specify requirements on a system (obligations policies). Such a mechanism is useful for the combination with the topic of Green IT. Policies have advantages like reusability, higher efficiency, expandability and better conflict management (Tonti et al., 2003) when it comes to managing large services or systems. Policies consist of rules that control the behavior of computer systems, derived from overarching goals such as company policy, service level agreements or safety requirements or for describing and enforcing e.g. energy efficiency requirements.

4 GREEN CLOUD SERVICES: SYSTEM AND ARCHITECTURE

To create a holistic solution that brings together model-based cloud services with Green IT policies end-to-end, we need to examine three things: service lifecycle, architectures of Cloud Management Platform and Policy Management Systems and their descriptor formats. The lifecycle of a model-based cloud service can be divided into four phases: modeling, deployment, instantiation and runtime. In the modeling phase, the TOSCA-modeler creates the topology template, plans, scripts, images and all other required artifacts required for the execution of the service and packages them into an archive (CSAR). In the next phase, the service definition and all related artifacts are stored in a database component, where they are made accessible by the public through e.g. a service catalog. Once published, the cloud service consumer can select a service from the service catalog - this triggers the instantiation of the service. The core component during the instantiation phase is the TOSCA-container, as it orchestrates the generation of the service, e.g. generating the virtual machines, installing the needed software, setting up the

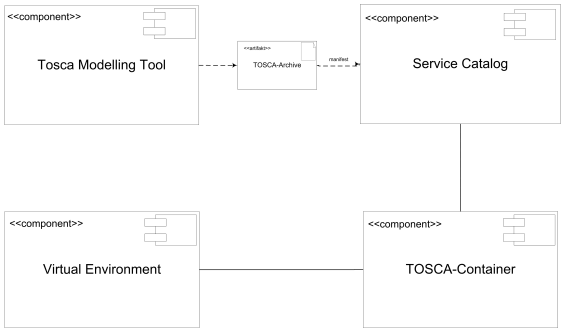

Figure 2: Overview of the components involved in a TOSCA Cloud Management Platform.

network connections etc. by running the associated plans which operate directly on the topology. Figure 2 gives an overview of the involved components.

The next step to create a holistic solution is to map the components from the Cloud Management Platform to key architectural components of a policy management system (IETF, 2001, OASIS, 2005). For illustration, we use the architecture model provided by the Internet Engineering Task Force (IETF, 2001). In this model, the basic components of a policy system are described - Figure 3 gives an overview.

The responsibility of the *Policy Management Tool* is the creation, modification and deployment of policies, the transformation from abstract to concrete policies as well as error handling. The *Policy Repository* provides a storage for the policies, as well as provides the policies when they should be provisioned. Finally, the *Policy Decision Point* interprets policies, receives events and determines the appropriate actions. These components match nicely to the pieces of a TOSCA cloud management platform. To illustrate and motivate the combination into a holistic system, we examine each lifecycle phase: In the *Modeling phase*, the TOSCA specification allows for the injection of policies at different points within a topology template. The policy is part of the topology template description and is stored within the CSAR. The modeler can either directly specify the policy code inside the topology template or refer to it by a reference (c.f.

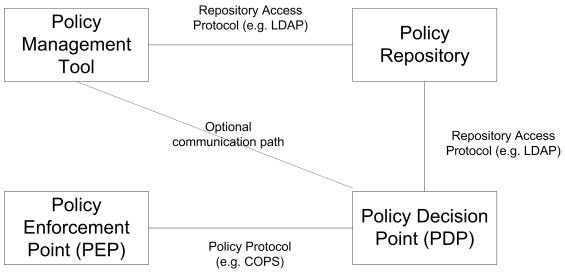

Figure 3: Overview of the components involved in a Policy System, according to (IETF, 2001).

section 5). The reference allows for the integration of externally defined policies. Thus, the TOSCA modeling supports creating and changing policies. During the modeling phase, all attributes of the policy tag have to be defined. The actual content of a policy is described within the tag. In the *Deployment phase*, the policy management tool component of the IETF-architecture is also responsible for the distribution of policies. In TOSCA, all artifacts including policies are grouped into an archive and stored in a service catalog, which is a implementation of the policy repository. When a service has been requested by a user, we enter the *Instantiation phase*. The CSAR archive with the artifacts is loaded into the TOSCA-container. In the receiving environment, the package is parsed and the individual artifacts are analyzed by a topology engine, a part of the TOSCA Container, which also realizes the order of the elements. The engine implements the topology template with the help of provisioning mechanisms into the production system.

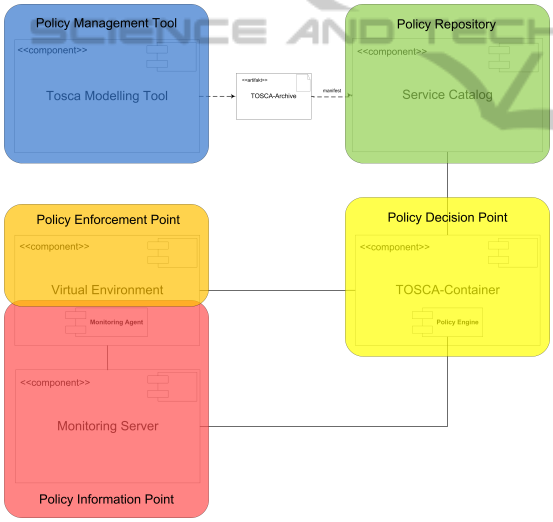

Figure 4: TOSCA- and Policy System Architectures combined.

To create an instance of an application, a *build plan* is executed. For the interpretation and execution of the plan, the TOSCA container uses a process engine which communicates with the hypervisor and triggers e.g., the creation of virtual machines. Additional software is installed on the virtual machines by starting installation scripts. The TOSCA Container picks the policy implementation based on the URI of the policy definition, but for the processing of the incoming events and to determine the actions to be performed, an additional policy engine is needed. It has to be able to afford all functions of the policy decision point. To respond to events, the policy engine includes an event handler component. The task of the event handler is to relate the events to the corresponding policies. Afterwards, the policies are interpreted, i.e., conditions are evaluated and the resulting instructions are determined. In the *Runtime phase*, the management and control of virtual machines is done by the hypervisor. The instructions, determined by the policy engine, will also be implemented by the hypervisor (policy enforcement point). Since virtual environments are very complex and include many properties, special monitoring agents are used to ensure a flexible and comprehensive monitoring. They are installed on the servers to monitor all relevant metrics. The considerations in this section have shown that it is possible to map the IETF- to the TOSCA- model. In Figure 4, the TOSCA-components and their equivalents are shown.

5 GREEN CLOUD SERVICES: METADATA FORMAT

After the combination of a TOSCA cloud management platform with a policy system has been outlined,

```
1  <ServiceTemplate ... >
2   <Import ... />
3   <TopologyTemplate>
4   <NodeTemplate id="VmApache" name="VM for
        Apache" type="ns1:VirtualMachine">
5   <Properties> ... </Properties>
6   <Policies>
7   <Policy
8   name="LowWorkloadPolicy"
9   policyRef="lamp:LowWorkloadPolicy-VmApache"
10  policyType="ns1:LowWorkloadPolicy"/>
11  </Policies>
12  </NodeTemplate>
13  <NodeTemplate id="VmMySQL" name="VM for
        MySQL" type="ns1:VirtualMachine">
14  <Properties> ... </Properties>
15  </NodeTemplate>
16  </TopologyTemplate>
17  <PolicyTypes>
18  <PolicyType
19  name="LowWorkloadPolicy"
20  targetNamespace="...">
21  <PropertiesDefinition element="ns1:
        LowWorkloadPolicy"/>
22  <AppliesTo>
23  <NodeTypeReference typeRef="ns1:
        VirtualMachine"/>
24  </AppliesTo>
25  </PolicyType>
26  </PolicyTypes>
27 </ServiceTemplate>
```

Figure 5: Excerpt of a service template describing an energy-efficient LAMP service.

let's illustrate input to such a holistic system based on the LAMP service template.

In TOSCA, a policy also follows the type / template pattern previously introduced: The *Policy Type* defines the properties of policy (e.g. the data needed to influence a behaviour), whereas *Policy Templates* provide actual values for the properties in a concrete scenario.

Figure 5 shows an excerpt from a Service Template for the LAMP service: In lines 4 and 13, two node templates have been defined, representing a WebServer and a Database. To emphasize our point, think of them as a Web Server Cluster and a Database Cluster. In line 19, a *LowWorkloadShutdownPolicy* has been attached to the Web Server Cluster. In Line 22, the payload of this policy is referenced.

The payload just defines the properties that an event is required to carry, in order for the policy definition point to evaluate the policy and trigger an appropriate action. In our example of shutting down a cluster member when the workload becomes too low, we need only two properties: the old and the new workload value. So far we did not define an action that needs to be taken, nor did we define a concrete policy language. Policies in TOSCA can be defined independently of a policy language. While this supports portability, it requires adaption to a specific language for every cloud provider. But TOSCA allows us also to provide a policy in a specific language:

```
1  <PolicyType  name="LowWorkloadPolicy"
2    policyLanguage="xs:http://www.ponder2.net/"
3    targetNamespace="...">

5   <PropertiesDefinition element="ns1:
        LowWorkloadPolicy"/>
6   <AppliesTo>
7    <NodeTypeReference typeRef="ns1:
         VirtualMachine"/>
8   </AppliesTo>
9  </PolicyType>
```

Figure 6: PolicyType from Figure 5 with a Policy Type defined in the language Ponder2.

In our prototype, we have chosen a policy language called *Ponder2* (PONDER, 2013). *Ponder2* is a declarative, object-oriented language that allows for the definition of both, obligation and authorization policies. *Ponder2* can be used in a wide range of different devices ranging from ad-hoc networks to distributed systems. A Ponder2 system provides the ability to manage a variety of objects such as sensors, alarm systems, routers, etc. These basic elements are called managed objects. They are created in Java and their provided functionalities can be adapted to the requirements of an environment and expanded accordingly. The standard distribution of Ponder2 includes three predefined object types: domains, events and policies. Additional functionality can be added to a system by means of new managed objects. To allow a dynamic interaction between heterogeneous IT-Environments, Ponder2 utilizes so called Self Managed Cells (SMC). SMCs represent independent systems, i.e. a set of hardware and software components that are able to manage themselves. The communication between the different components occurs through an content-based event bus. For the message exchange, the proprietary language PonderTalk is used. The Java objects implement an interface (Managed Object) that enables the mapping from PonderTalk to Java methods. For the communication between different systems, the Ponder2-Framework offers the possibility to access external Managed Objects. Thus, objects can be exported and imported between different SMCs.

To make the holistic system work, leveraging the *Ponder2* technology, several steps are necessary: a.) An event emitter has to be installed on the virtual machines - in our example, both on the mySQL VM and on the WebServer VM. Conceptually, it is an agent that monitors relevant metrics from the VMs (e.g., the CPU load) and triggers respective events. This design is very much in line with industrial cloud management systems, where a set of agents (for various purposes) are installed on top of the provisioned VMs. b.) The produced event has to be send to the TOSCA-container where an obligation policy interpreter receives the event and looks for applicable policies. This requires c.) correlating the event to a running instance and then identifying the model where this instance was derived from to get the policy. The policy then evaluate their conditions and invoke the corresponding actions (Keoh et al., 2007). Figure 7 illustrates how the concrete Ponder2 code looks like:

```
1  event:= root/factory/event create: #("name" "
       new" "old").
2  root/event at: "greenEvent" put: event.
3  policy:= root/factory/ecapolicy create.
4  policy event: root/event/greenEvent.
5  policy condition: [ :name :oldValue :newValue
       |
6  name == "Workload" & (new < 30) & (old >= 30)
       ].
7  policy action: [
8  // Execute shutdownClusterMember Plan to
       shutdown a cluster member].
9  root/policy at: "workloadLow" put: policy.
10 policy active: true.
```

Figure 7: Concrete LowWorkloadPolicy in PONDER2.

In Figure 7, a management plan called *shutdownClusterMember* is triggered by the policy, following the event-condition-action (ECA) rule used in policy systems. Figure 8 illustrates the plan: First the plan has to check whether the cluster manager is available - obviously, this is a safety measure. If successful, the plan issues a request against the cluster member to shut down. This step might require more detail, e.g., if it is a database we might want to stop the datasource first, to avoid data inconsistencies, but conceptually this is the step that needs to be taken. Once stopped, the cluster member can be removed from the cluster manager and thus will no longer consume energy.

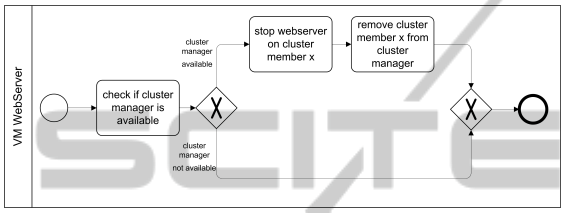

Figure 8: The *shutdownClusterMember* Plan.

A policy-based mechanism to shut down a cluster member to save energy is just one example how policies can be used to realize energy-efficient cloud services. Other use cases can be imaged: If clouds would have an energy certification, i.e. the data center is operated on renewable energy, a cloud service could declare a policy requiring such a certification, and if not fulfilled the service model is not allowed to be instantiated. Furthermore, policy-based scheduling or workload planning or other energy-saving, cloud service specific measures can be imagined.

6 CONCLUSION

In this paper, we have shown how to design, develop and run standardized, energy-efficient cloud services. We extended a TOSCA cloud management platform with policy handling parts. We showed how to combine TOSCA models with policies (in the policy language *Ponder2*). Thereby we realized description and execution of portable green cloud services. Ponder2 has been chosen as our policy language because of its dynamicity and versatility. We believe this work to be the foundation to combine two of the most important future markets of the IT industry. To our knowledge there is no work conducted to combine cloud services with Green IT aspects so far, so this work to combine cloud services with ecological aspects promises extraordinary potential for cloud service providers. With cloud computing in general becoming more and more a commodity, the offering of energy efficient services will become a competitive advantage as well as it will serve the environment.

This work was partially funded by the BMWi project Migrate! (01ME11055).

REFERENCES

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing.

Berl, A., Gelenbe, E., Girolamo, M. D., Giuliani, G., Meer, H. D., Dang, M. Q., and Pentikousis, K. (2010). Energy-efficient cloud computing.

Buyya, R., Beloglazov, A., and Abawajy, J. (2010). Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*.

Dargie, W. (2012). Analysis of the power consumption of a multimedia server under different dvfs policies. In *CLOUD*. IEEE.

Duy, T. V. T., Sato, Y., and Inoguchi, Y. (2010). Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum*.

Hwang, C.-H. and Wu, A. (2000). A predictive system shutdown method for energy saving of event-driven computation.

IETF (2001). Ietf policy model. http://tools.ietf.org/html/rfc3060.

Keoh, S. L., Twidle, K., Pryce, N., Schaeffer-Filho, A. E., Lupu, E., Dulay, N., Sloman, M., Heeps, S., Strowes, S., Sventek, J., and Katsiri, E. (2007). Policy-based management for body-sensor networks. In *4th Int. Workshop on Wearable and Implantable Body Sensor Networks*. Springer Berlin Heidelberg.

Liu, Z., Lin, M., Wierman, A., Low, S., and Andrew, L. (2011). Geographical load balancing with renewables.

OASIS (2005). Xacml policy model extensible access control markup language (xacml) version 2.0, pp. 16-18.

OASIS (2013). Tosca - topology and orchestration specification for cloud application.

PONDER (2013). http://www.ponder2.net.

Sueur, E. L. and Heiser, G. (2010). Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proc. of the 2010 int. conf. on Power aware computing and systems*. USENIX Association.

Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., and Uszok, A. (2003). Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *The Semantic Web-ISWC 2003*. Springer Berlin Heidelberg.

Wang, L., von Laszewski, G., Dayal, J., and Wang, F. (2010). Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *CCGrid*. IEEE/ACM.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Younge, A. J., von Laszewski, G., Wang, L., Lopez-Alarcon, S., and Carithers, W. (2010). Efficient resource management for cloud computing environments. In *Green Computing Conf.* IEEE.

Zhang, Y., Wang, Y., and Wang, X. (2011). Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware 2011*.