Armstrong *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2017) 6:14
DOI 10.1186/s13677-017-0083-2

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Towards energy aware cloud computing application construction

Django Armstrong, Karim Djemame* and Richard Kavanagh

Abstract

The energy consumption of cloud computing continues to be an area of significant concern as data center growth continues to increase. This paper reports on an energy efficient interoperable cloud architecture realised as a cloud toolbox that focuses on reducing the energy consumption of cloud applications holistically across all deployment models. The architecture supports energy efficiency at service construction, deployment and operation. We discuss our practical experience during implementation of an architectural component, the Virtual Machine Image Constructor (VMIC), required to facilitate construction of energy aware cloud applications. We carry out a performance evaluation of the component on a cloud testbed. The results show the performance of Virtual Machine construction, primarily limited by available I/O, to be adequate for agile, energy aware software development. We conclude that the implementation of the VMIC is feasible, incurs minimal performance overhead comparatively to the time taken by other aspects of the cloud application construction life-cycle, and make recommendations on enhancing its performance.

Keywords: Cloud computing, Virtualization, Energy efficiency, Cloud engineering, Cloud architectures, Cloud interoperability, Performance evaluation

Introduction

Current trends in industry show continuous growth in the adoption and market value of cloud computing with many companies changing their business models and products to adapt to a service oriented outlook. Cloud computing as a leading ICT approach provides elastic and on-demand ICT infrastructures makes up a large proportion of the total ICT energy consumption. Predictions have been made on an unsustainable quadrupling in the energy consumption and carbon emissions of data centres used to operate cloud services by 2020 [1] with comparable emissions to the aeronautical industry. As energy efficiency is at the heart of governments/institutions for smart, sustainable and inclusive growth as part of a transition to a resource efficient economy, considering and improving the energy efficiency of cloud computing is therefore of paramount importance.

Research on energy efficiency in clouds has attracted considerable attention and has focused on many aspects including ICT equipment (servers, networks) as well as software solutions running on top of ICT equipment (e.g. cloud management system domain for managing the cloud infrastructure), see [2] for a survey. This paper is concerned with the topical issue of energy efficiency in clouds and specifically focuses on the design and construction of cloud services through the implementation of tools within a reference energy-aware and self-adaptive architecture. Such architecture provides novel methods and tools to support software developers aiming at optimising energy efficiency and minimising the carbon footprint resulting from designing, developing, deploying and running software in clouds. Cloud services are made of several shared software components, which are utilised many times. These components can then be characterised, which allows the Software developers to relate service construction and energy use. This relationship will further depend on the deployment conditions and the correct operation of the service, which can be achieved by means of an adaptive environment.

Software developers need to construct and analyse their applications using a programming model as part of a Software as a Service (SaaS) cloud layer. Currently, they usually optimise code to achieve high performance but

*Correspondence: K.Djemame@leeds.ac.uk
School of Computing, University of Leeds, Leeds, UK

guidelines for energy optimisation are also valuable. Consider an application which is developed using a programming model and the resultant program is analysed for potential energy hotspots [3]. Methods from automatic complexity analysis and worst-case execution time analysis of the application can be extended and combined with energy models of the hardware, giving the developers an approximate energy profile for the program. This analysis provides feedback to the software developers thus enabling an adaptive software development methodology through monitoring (the code's performance), analysing (identifying energy hotspots), planning (potential changes to the code to improve its performance) and executing (recompiling the code enabling further monitoring).

Similarly for software developers evaluating different deployment scenarios for the applications will need various installation configurations. For example, a developer models via UML the different deployment scenarios they wish to evaluate for potential use in a production environment. After which, these models are translated into descriptions, e.g. XML that can be processed into deployable virtual format artefacts.

In both cases *image construction* is required prior to the application deployment scenario which is realised as a collection of Virtual Machines (VMs) containing application components. To the best of our knowledge, no current software solution provides capabilities to both generate base images that contain a functional operating system and install and configure a cloud application automatically. This therefore provides the opportunity to create multiple different configurations of an application, where these different deployment configurations can be exploited, by selecting the most appropriate configuration to serve the required system load while saving energy. For comparison, a tool such as Packer [4] can be used to create golden images for multiple platforms from a single source configuration but does not provide support for the automated installation of software into these images. Another example tool such as Vagrant [5] enables software development teams to create identical development environments but does not provide a mechanism to automate the deployment of software into these environments.

A Virtual Machine Image Constructor (VMIC) is therefore key to support adaptive software development processes in an energy-aware SaaS architecture. Such component implements the automation of image construction that would otherwise make the burden and cost too high of considering iteratively adapting an application to use less energy in the software development stage through incremental out-of-band (of normal application operation) test based deployment scenarios. In addition to this, the VMIC is considered as an important contribution from the perspective of Software Engineering filling the gap between generating base images in cloud computing and automatic configuration of cloud applications. This component supports the energy efficiency goal within the cloud architecture by providing means of packaging cloud applications in a way that enables provider agnostic deployment, while maintaining energy awareness.

The paper's main contributions are:

1. The detailed architecture of a SaaS layer component that facilitates an energy aware and efficient cloud development methodology.
2. The results of a performance evaluation and feasibility study of the VMIC tool for the construction of cloud application components.
3. Our practical experience during implementation and recommendations on enhancing the performance of the tool.

The remainder of the paper is structured as follows: "Related work" section reviews the literature on energy-aware cloud computing. "Energy efficient cloud architecture" section describes the proposed architecture to support energy-awareness with emphasis on the importance of the SaaS layer for facilitating energy efficiency in cloud applications. "Cloud engineering" section explains our vision of a self-adaptive development life-cycle and how this can enable energy aware cloud application construction through our VMIC tool. "Experimental design" section presents the experimental design where we evaluate the performance of the VMIC tool, and "Results and recommendations" section discusses the results. Finally, "Conclusion" section provides a conclusion on this paper and discusses plans for future work.

Related work

Research effort has targeted energy efficiency support at various stages of the cloud service lifecycle. In the *service development stage*, *requirements elicitation* includes techniques for capturing, modelling and reasoning with energy requirements as well as product line oriented techniques to model and reason about system configuration [6, 7]. In terms of *software design* in relation to energy consumption, some research efforts relate energy awareness and optimization at the application and system level [8], focus on profiling the application's energy consumption at runtime to iteratively narrow down on energy hot spots [9], or considers cloud architecture patterns to achieve greener business processes [10]. Energy efficiency has also been the subject of investigation in *Software development*, e.g. by studying the energy consumption of the application prior to deployment [11]. In the service *deployment stage*, research effort has focused on *Service Level Agreement* (SLA) deployment strategies especially with regard to SLAs that are energy-aware, e.g. by implementing specific policies to save energy [12, 13], as well

as service deployment technologies which play a critical role in the management of the cloud infrastructure and thus have an effect on its overall energy consumption [14]. In the service *operation* stage, energy efficiency has been extensively studied and has focused for example on approaches towards energy management for distributed management of VMs in cloud infrastructures, where the goal is to improve the utilization of computing resources and reduce energy consumption under workload independent quality of service constraints [15].

Configuration management tools provide four core benefits to managing the cloud. These are i) the reproducibility and industrialization of software configuration, ii) the continuous vigilance over running systems with automated repairs and alert mechanisms, iii) enhanced control over and rationalisation of large scale deployments and iv) the ability to build up a knowledge base to document and trace the history of a system as it evolves. The most well know tools include CFEngine, Puppet and Chef.

CFEngine [16] provides automated configuration management for large networked systems and can be deployed to manage various infrastructure such as servers, desktops and mobile/embedded devices. It uses decentralised, autonomous software agents to monitor, repair and update individual machines. CFEngine central concept is the idea of convergence [17], where the final desired state of the system is described instead of the steps needed to get there. This enables CFEngine to run whatever the initial state of the system is with predictable end results. The downside of this approach is that only statistical compliance or best effort can be achieved with a given configuration policy, where by a system cannot be guaranteed to end up at a desired state but slowly converges at a rate defined by the ratio of environmental change to the rate at which CFEngine executes. Puppet [18] was forked from CFEngine and provides graph and model driven approaches to configuration management, through a simplified declarative domain specific language that was designed to be human readable.

Chef [19], a fork of Puppet, places emphasis on starting up services from newly provisioned clean systems, where the sequence and execution of configuration tasks is fixed and known by the user. This makes Chef particularly well suited to the paradigm of cloud computing where VM instances are short lived and new instances are spawned from a newly provisioned base image. Chef uses the analogy of cooking and creates "recipes" that are bundles of installation steps or scripts to be executed.

The Open Virtualization Format (OVF) is an open standard for defining, packaging and distributing virtual appliances that can run virtualized on a cloud [20]. Its use as part of a service descriptor to define the requirements of an applications is not new and has been implemented within the OPTIMIS Toolkit [21], where an OVF fragment resides in a non-standard XML based service manifest schema. One issue with this approach is the impact on interoperability with cloud providers that need to support this schema to enable application deployment. This compared to the solution that is presented in this paper where a pure OVF document is used, extended and implemented according to the capabilities of the OVF Specification version 1.1.1, makes our solution 100% compliant with cloud providers and technologies that already support OVF.

In this paper, the proposed software tool provides capabilities to both generate base images that contain a functional operating system and install and configure a cloud application automatically. This sits alongside previous work [14, 22] that allows for the contextualisation and recontextualisation of virtual machines, so that the environment can be reconfigured dynamically at runtime, leading to a dynamic reconfigurable environment. This is key to support: 1) adaptive software development processes in a cloud architecture, and 2) the energy efficiency goal within the architecture by providing means of packaging cloud applications in a way that enables provider agnostic deployment, while maintaining energy awareness.

Collectively the automated configuration and reconfiguration of cloud applications, along with enhanced energy awareness of the different deployment solutions gives rise to the possibility of performing energy saving techniques. These techniques can be quite expansive such as: consolidation [23], horizontal and vertical scaling and the usage of DVFS [24] and RAPL [25].

Energy efficient cloud architecture

To reduce the energy consumption of a cloud system, a reference architecture is needed to first enable energy awareness of all phases of an application's life-cycle and secondly provide actuators to reduce and optimizes energy efficiency. To this end we have realised such a reference architecture through the implementation of a toolbox, details of which can be found in [26]. To facilitate the reader's understanding of the research in this paper, Figs. 1, 2 and 3 provide an overview of the proposed architecture which includes the high-level interactions of all components, is separated into three distinct layers and follows the standard cloud deployment model.

In the SaaS layer, illustrated by Fig. 1, a set of components and tools interact to facilitate the modelling, design and construction of a cloud application. The components aid in evaluating energy consumption of a cloud application during its construction. A number of plug-ins are provided for a frontend *Integrated Development Environment* (IDE) as a means for developers to interact with components within this layer. The Requirements and Design Modelling Plug-in provides developers with tools to aid in is based on the energy aware modelling

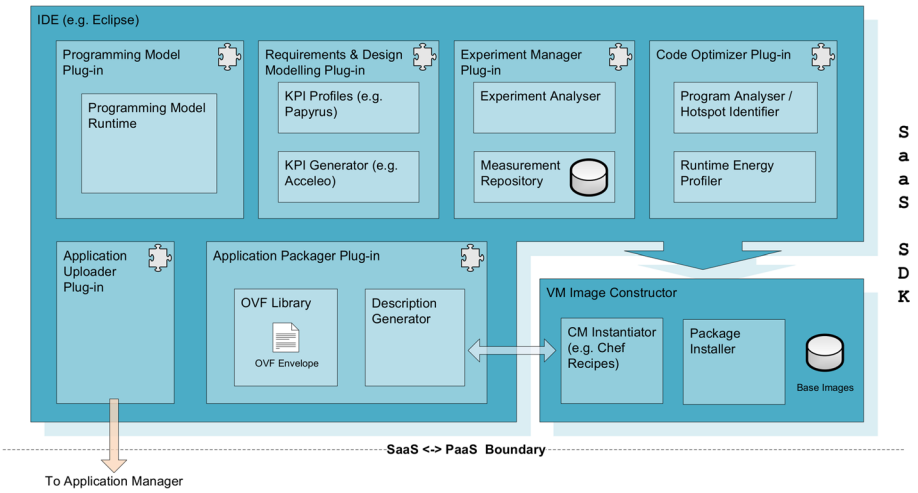

Fig. 1 Architecture - SaaS

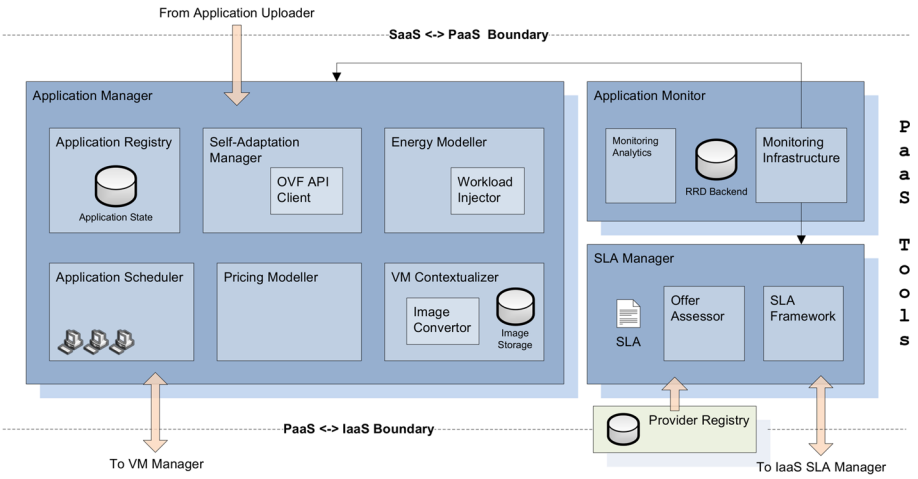

Fig. 2 Architecture - PaaS

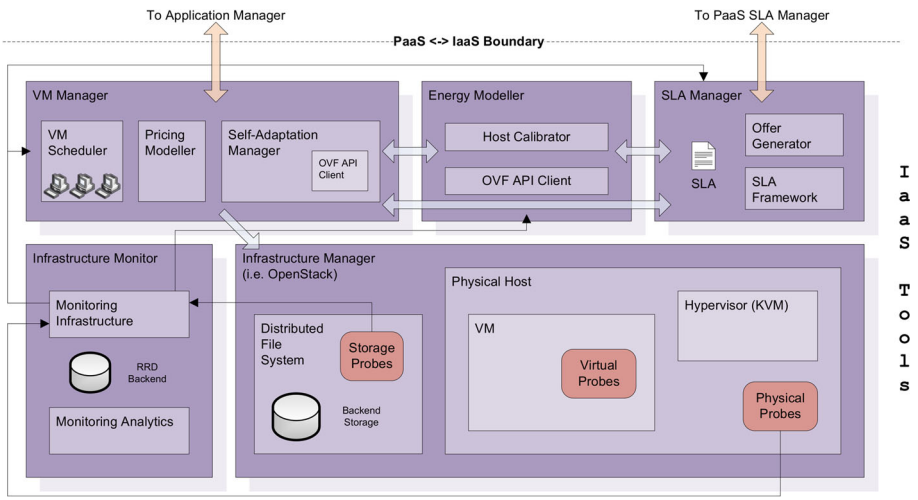

Fig. 3 Architecture - IaaS

of an application, while the Code Optimiser Plug-in provides offline functionality to profiler an application's energy consumption during development. The Programming Model Plug-in is based on COMPSs [27] and provides an interface to the developer to create applications that follow the energy aware programming model [28]. Finally the Deployment Experiment Manager Plug-in, helps to associate the outputs of the SaaS Modelling tools with the workloads and the energy-aware architecture.

A number of packaging components and tools are also made available to enable cloud provider agnostic deployment of the constructed cloud application, while also maintaining energy awareness. The Virtual Machine Image Constructor (VMIC) tool is responsible for fabricating the images and included software needed to deploy an application, which is in turn handled by the Application Uploader Plug-in, which uploads a packaged application created in the SaaS SDK layer to the Application Manager at the PaaS layer. The VM Image Constructor communicates with both the Programming Model Plug-in and Application Packager Plug-in for the installation of the programming model runtime and for packaging more traditional cloud applications into base images.

The PaaS layer, illustrated by Fig. 2, provides middleware functionality for a cloud application and facilitates the deployment and operation of the application as a whole. Components within this layer are responsible for selecting the most energy appropriate provider for a given set of energy requirements, stored as OVF properties, and tailoring the application to the selected provider's hardware environment. Application level monitoring is also accommodated for here, in addition to support for Service Level Agreement (SLA) negotiation.

In the IaaS layer, illustrated by Fig. 3, the admission, allocation and management of virtual resource are performed through the orchestration of a number of components. The Virtual Machine Manager (VMM) is responsible for managing the complete life cycle of the VMs that are deployed in a specific infrastructure provider. Energy consumption is monitored, estimated and optimized using translated PaaS level metrics. These metrics are gathered via a monitoring infrastructure and a number of software probes. The *Energy Awareness* provision is an important step in the architecture implementation plan as it concentrates on delivering energy awareness in all system components. Monitoring and metrics information is measured at IaaS level and propagated through the various layers of the cloud stack (PaaS, SaaS) via the use of a OVF document.

The *Cloud Stack Adaptation* with regard to energy efficiency focuses on the addition of capabilities required to achieve dynamic energy management per each of the cloud layers, in other words:

1. *Intra* layer adaptation: refers to local layer adaptation in isolation. It considers the extensions of the runtime environment in order to be able to orchestrate the invocation of different application components with advanced scheduling techniques that take into account energy efficiency parameters.
2. *Inter* layer adaptation: the aim is to achieve steering information among cloud layers for triggering other layers to adapt their energy mode, the focus being information sharing and decision making among SaaS, PaaS and IaaS layers.

The key research challenge is the ability to take adaptive actions based upon energy consumption, performance and cost factors within each layer of the architecture and examine the effect that these have upon the running applications. Self-adaptive cloud-based software applications can be realized via a MAPE (Monitor, Analyse, Plan and Execute) feedback control loop architecture [29]. The cloud stack adaptation is then tailored for the Self-Adaptation Manager that manages applications at runtime and maintains performance and energy efficiency at the PaaS (Fig. 2) and IaaS (Fig. 3) layers, respectively. This is the subject of continued work [30].

Cloud engineering

Cloud engineering plays an important role in the context of creating energy efficient application development. Automation of cloud engineering tasks such as out of band testing and automated deployment are also necessary to gain full benefits from a cloud provider. In this section we discuss the importance of a self-adaptive development life-cycle that considers application energy consumption while maintaining other more traditional quality aspects of software such as performance at an acceptable level.

Towards a self-adaptive development life-cycle

To enable energy awareness in cloud applications, a self-adaptive software development methodology that considers energy at each stage: requirements gathering, software construction and testing is a necessity. Self-adaptation in this context refers to the ability to provide feedback in the form of energy metrics that guide a developer within an iterative development process towards an optimal energy efficient software solution. For any such methodology to exist a number of tools must be available to reduce the burden of energy consumption optimization. This is where the SaaS layer components of the architecture presented in 'Energy efficient cloud architecture" section can reduce time-to-market of cloud applications through the automation of standard developer practices, enabling quicker feedback on energy efficiency and more development iterations. Some of the more time consuming and challenging aspects of developing a cloud Application

are the construction of VM images and installation of software dependencies. This is where the work on the VMIC is positioned. This tool manages software dependency installation and creates cloud provider agnostic VM images in an automated fashion, reducing developer effort.

Cloud application construction - VMIC

Figure 4 shows the automated interaction between the VMIC and other components in the SaaS layer of the energy efficient architecture. The VM image construction process is coordinated by Eclipse Plug-ins which invokes the VMIC with an application description. This description is provided in the form of an industry standard OVF [20] document. The VMIC parses the application descriptor for details on what packages should be installed in each base image, where each image represents a packaged software component to form a cloud application.

The VMIC can operate in two modes, *offline* and *online*. The first phase of the VMIC automation process copies an appropriate base image such as a variant of Linux or Windows. After copying, depending on the nature of the packages to be installed, it either mounts the image using `qemu-nbd` [31] (*offline* mode) or instantiates it remotely (*online* mode) using `libvirt` [32]. During *offline* mode packages are placed into the Web root of a pre-installed tomcat container and is primarily used by the Programming Model Plug-in. In *online* mode, a locally running Chef [19] configuration management server is used to issue Chef recipes to the base VM that contains configuration information, package repositories and packages to install. Finally after packages have been installed in the VM, the image is saved either by unmounting the image, as is the case with *offline* mode or by saving a snapshot of the base VM image to the local file system, as is the case in *online* mode. The image and its content can then be tested on a local infrastructure or passed to the PaaS layer for future deployment on an IaaS provider.

These two modes of operation cover the necessary features and functionality to enable support for both COMPSs [27] enabled applications (packaged as Java Tomcat Web Services) invoked via the Programming Model Plug-in and more generic cloud applications (such as a n-tier web application) invoked via the Application Packager Plug-in, while minimising the time to create an

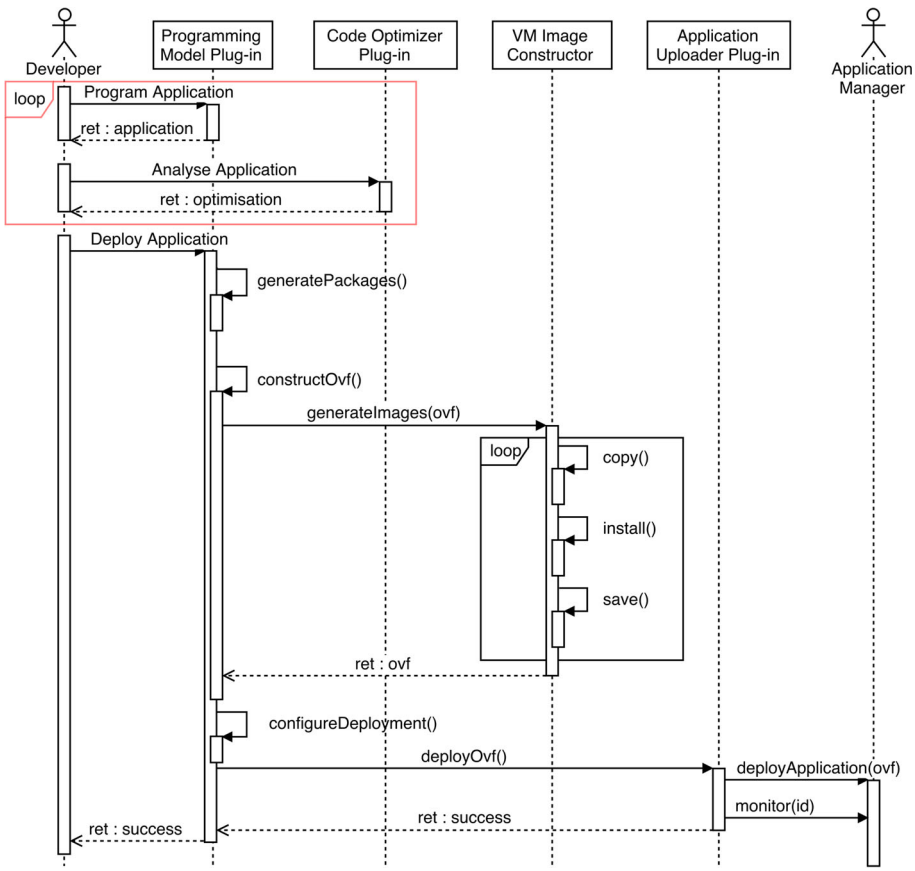

Fig. 4 SaaS Sequence Diagram

image. The remainder of this paper concentrates on the more flexible and challenging *online* mode.

Figure 5 illustrates the subcomponent that comprise to make up the VMIC in addition to showing interactions with external components and baseline technologies. The VMIC is comprised of two main Java packages one for each mode of operation and a Java library for parsing OVF. The process of constructing an image involves a number of different internal phases and coordination of baseline technologies within the VMIC. These phases are highlighted in order as follows:

1. **Initialise** - The OVF document passed to the VMIC is parsed for VM components attributes and their associated Chef *cookbooks* that contain *recipes* (instructions for installing software written in a domain specific language).
2. **Copy Image** - For every component and the given image type parsed from the OVF, the VMIC selects and copies an appropriate base image.
3. **Boot Image** - Using libvirt the baseline image selected is instantiated via a hypervisor such as KVM [33].
4. **Bootstrap Image** - Once the instantiated VM's operating system has initialised, the VMIC bootstraps a Chef Client either via Secure Shell (Linux) or Windows Remote Management, registering it with a local Chef server.
5. **Upload Cookbooks** - The VMIC downloads *cookbooks* from URLs parsed from the OVF that reside on a remote repository running on a Web server. These cookbooks are then uploaded to the local Chef Server.
6. **Deploy Cookbooks** - The VMIC associates the uploaded Chef cookbooks with the instantiated VM within the Chef Server and invokes the Chef Client to download and install the cookbooks.
7. **Clean Up** - After installation is complete, the VMIC shuts down the instantiated VM and deletes the uploaded *cookbooks.*

Experimental design

To evaluate the feasibility of the VMIC architecture and the construction phases as outlined previously, the following section discusses the experimental design to test the performance of the VMIC tool. "Cloud testbed"

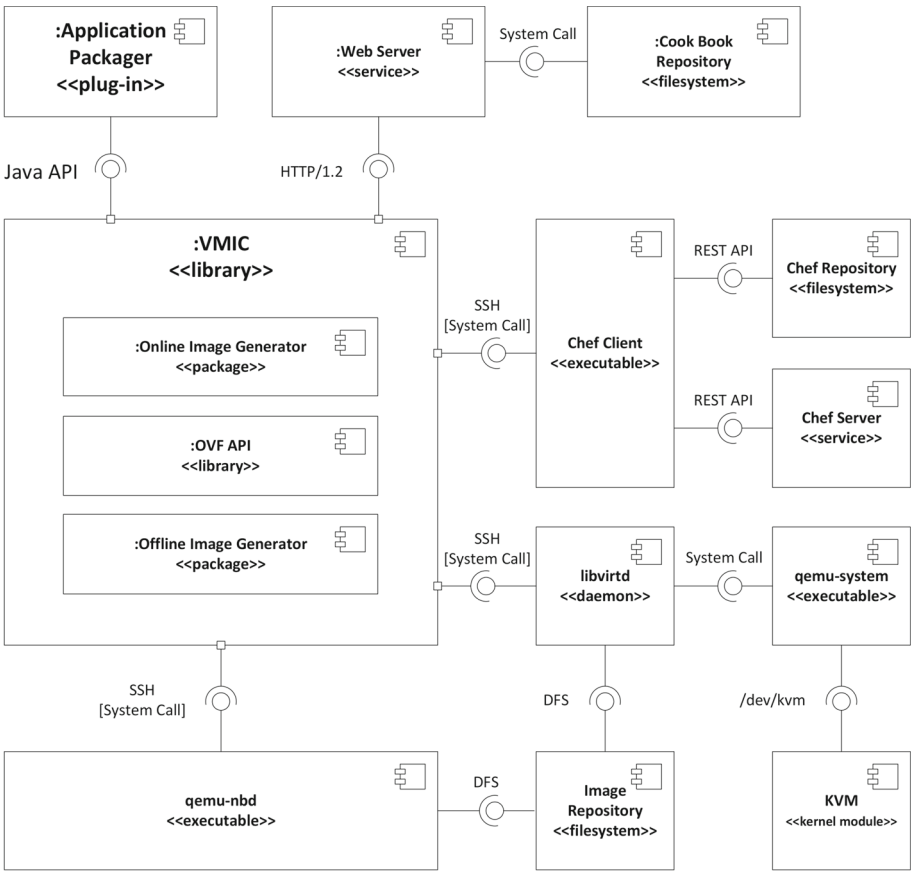

Fig. 5 VMIC Component Diagram

subsection discusses the cloud testbed used for the experimentation and the environment in which the VMIC was deployed. "Cloud application & experimental set-up" subsection describes the cloud application used to test the VMIC and the experimental set-up that includes a description of variables monitored.

Cloud testbed

The cloud testbed used in experimentation is located at the *Technische Universität Berlin* (see Fig. 6). The computing cluster consists of 200 commodity 1U nodes and 4 2U nodes used as a staging environment. Each of the 1U nodes is equipped with a quad-core Intel E3-1230 processor at 3.3 GHz, 16 GB of RAM, 1 TB of local hard disk capacity. The 2U nodes are equipped with 2 quad-core Intel E5620 processors at 2.66 GHz, 32 GB of RAM, 750 GB of local hard disk capacity. An 8 node Ceph cluster with a replica pool size of 2 and 16 TB of usable storage provided a Distributed File System (DFS). Additionally, the cloud testbed deploys OpenStack [34] to manage virtual infrastructure and Zabbix [35] to store monitored data. The power measurements were taken with *Gembird EnerGenie Energy Meters* [36]. All experiments presented in this paper were performed on the 1U nodes.

Each node is connected to two different networks (See Fig. 7) and is able to transfer in duplex at full speed 1 Gbit/s. The first network is dedicated for infrastructure management via OpenStack, as well as regular data exchange between the nodes and VMs (both private and public subnets). The second network is available for storage area network usage only, with storage nodes accessible via the Ceph DFS. Access to the testbed is provided by VPN.

Figure 8 illustrates the deployment configuration of the VMIC in the context of the energy efficient cloud architecture (see "Energy efficient cloud architecture" section) as implemented by the ASCETiC project [37]. The ASCETiC components/tools were deployed by layer into three VMs:

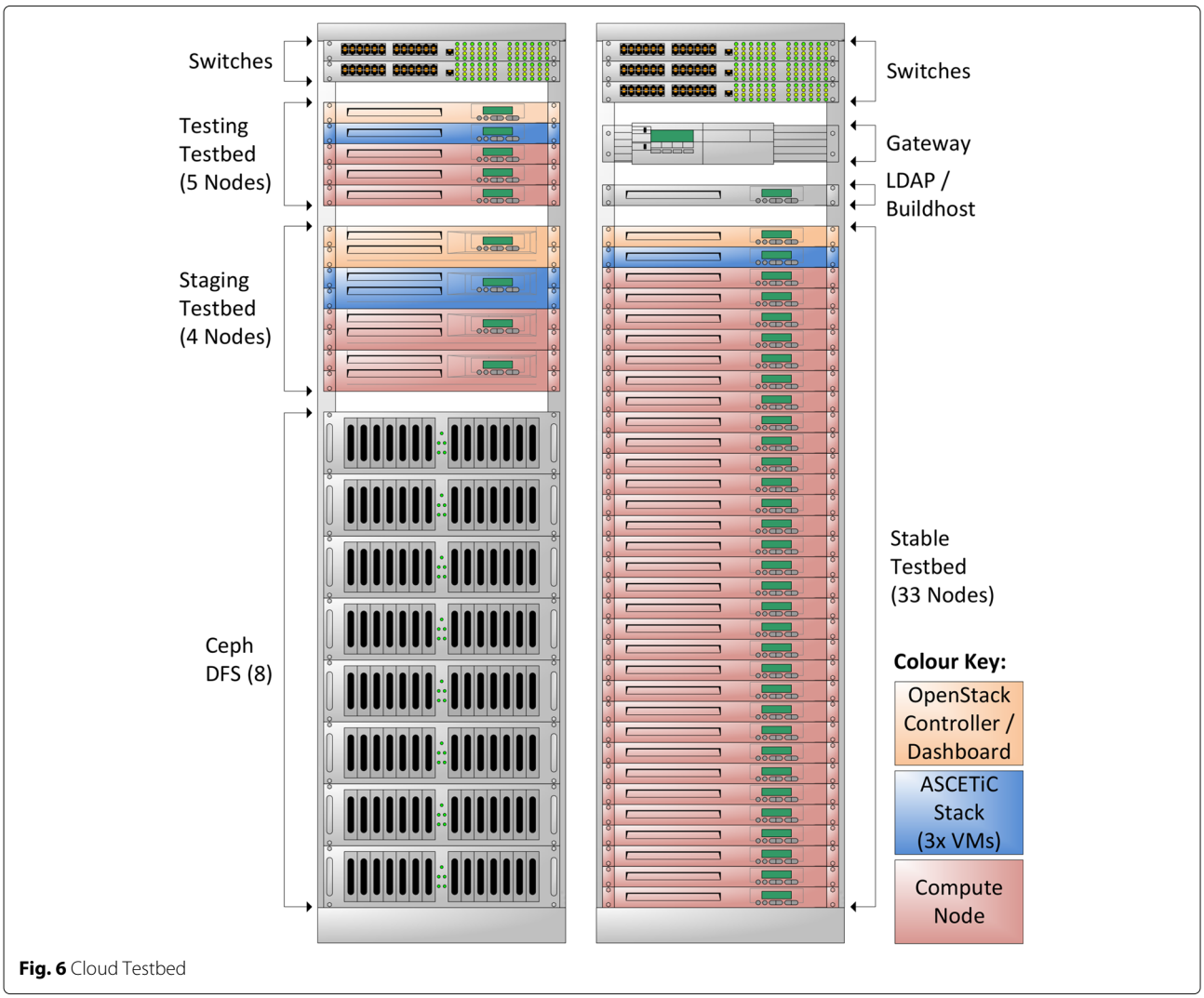

Fig. 6 Cloud Testbed

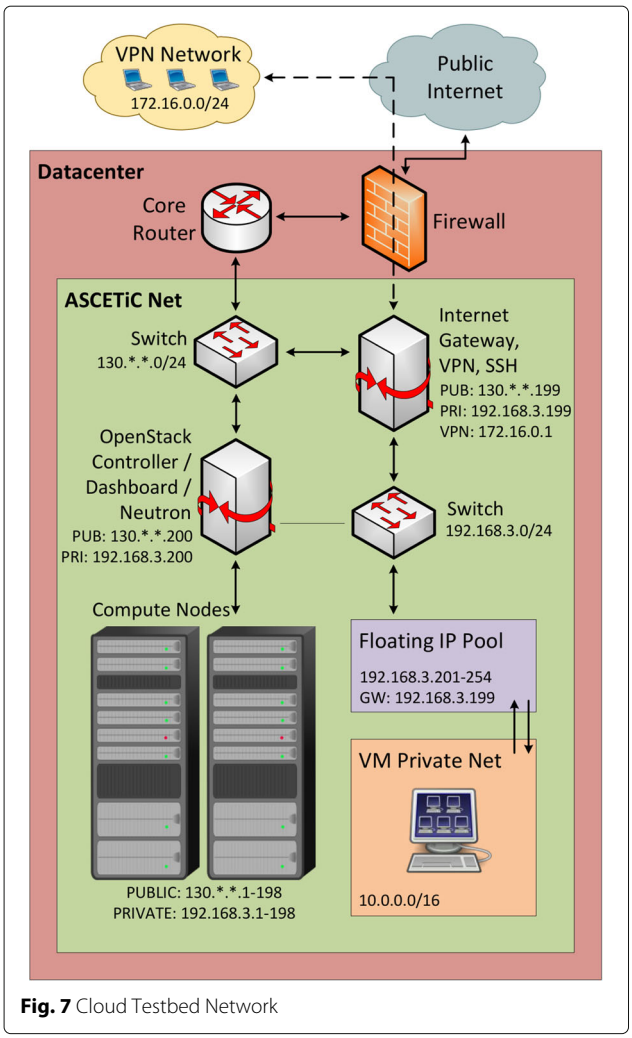

Fig. 7 Cloud Testbed Network

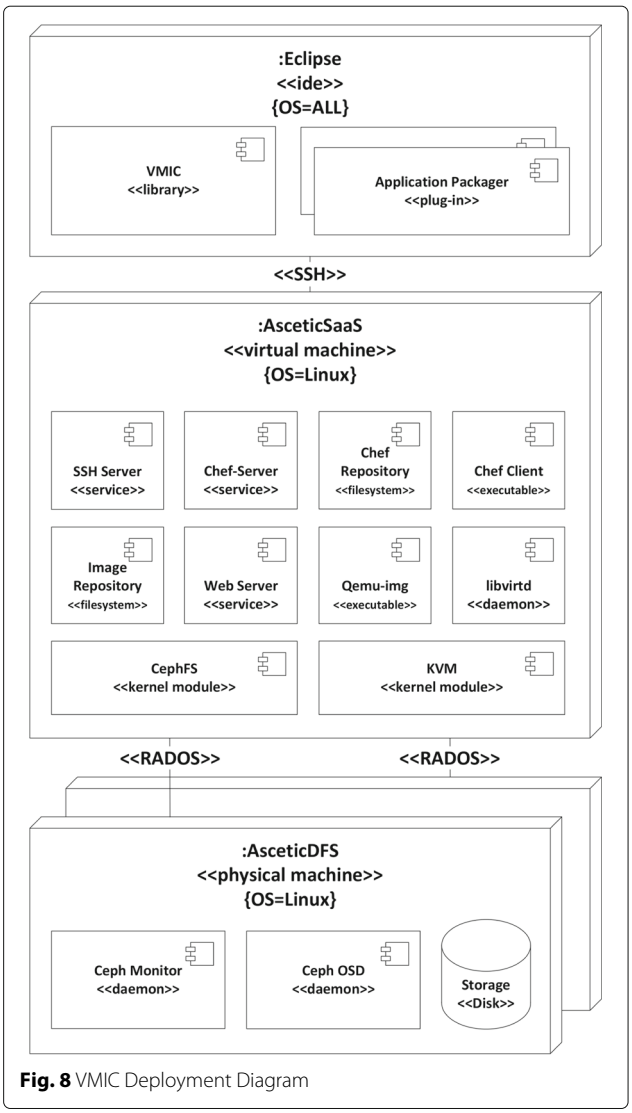

Fig. 8 VMIC Deployment Diagram

SaaS, PaaS and IaaS. The VMIC and its associated baseline technology dependencies were installed within the SaaS VM along with an instance of Eclipse containing the toolbox plug-ins. Nested KVM virtualization was used to enable the VMIC to create VMS within the SaaS VM and the nested VMs were backed by a mounted Ceph storage pool accessed via the `cephfs` Linux kernel module. The Ubuntu 12.04.5 LTS operating system was installed in each VM and ran on the Linux Kernel version 4.5.2. The SaaS VM was allocated 4 CPU cores and 8 Gbyte of memory from a 1U node.

Cloud application & experimental set-up

The chosen application to fulfil the objectives of the experiments is the NewsAsset [38] application that facilitates digital journalism. This is an n-tier application that is composed of a set of VM images as illustrated in Fig. 9. The application has a (fat) client in the front-end, a NewsAsset server implementing the business logic (in the middle layer) and stores news items data in an Oracle database in the back-end [39].

The load balancer image implemented via HAProxy [40] distributes load between NewsAsset application servers and was installed in a Linux Ubuntu 12.04.5 LTS base image. These application servers running on Windows operating system are comprised of a single binary executable and associated software dependencies. The NewsAsset application stores and retrieves data within a single Oracle database image and single File Server image, both running on Windows. Each VM is allocated 2 CPU cores and 2 Gbytes of memory. Table 1 outlines the software dependencies of each VM image as specified in the OVF passed to the VMIC.

To ascertain the performance of the VMIC when constructing the images for the NewsAsset application four experiments are performed:

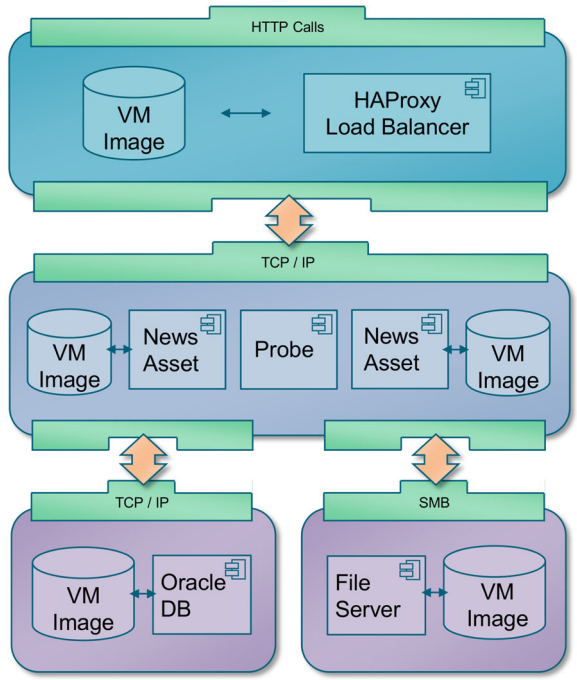

Fig. 9 Architecture of NewsAsset Application

1. The first evaluates the performance of each VMIC phase through the construction of the single image containing the application server component, the independent variable. Dependent variables in this experiment are the fine grained resource utilities of the ASCETiC SaaS VM: CPU utility and network bandwidth.
2. The second evaluates the runtime performance (execution time) of the VMIC. Linux and Windows components are compared, in addition to the time to generate all News Asset images over 10 consecutive runs.
3. In the third, the runtime performance of the VMIC is evaluated with a range $(1-4)$ of concurrently generated HAProxy Linux based images again over 10 consecutive runs.
4. Finally, the fourth demonstrates the VMIC's capabilities to aid in the deployment of NewsAsset energy aware application thanks to monitoring its power consumption.

Results and recommendations

The following section discusses the performance of the VMIC by presenting an analysis of the experimental results and illustrating adequate performance of its tool implementation.

Figure 10 demonstrates the performance characteristics of each VMIC phase. The graphs show the majority of time within the VMIC is spent in three phases. Firstly, the deployment of the cookbooks accounts for nearly 50% of the execution time. Secondly, the copy of the base image accounts for 20% of the execution time and is a factor of the size of the base image. Finally, the bootstrap process that sees the installation of the Chef client accounts for 10% of the runtime and the other phases tally for the remaining 30%. It is worth noting that a very slight delay at the start of the trace before "Copy Image" shows the minimal overhead induced from processing OVF, in the order of 1% of the total runtime.

Reviewing the network bandwidth consumption of the VMIC reveals that the copy of the base image is restricted by the available disk bandwidth over the network (a limitation of the testbed's 1 Gbit network to the Ceph cluster),

Table 1 NewsAsset application software dependencies

	Dependencies		
Component	1	2	3
HAProxy	*cpu 0.2.0*	*build-essential 6.0.0*	*haproxy 1.6.7*
Description	Chef cookbook to manage CPU related actions on linux.	Installs packages required for compiling C software from source.	Installs haproxy and prepares the configuration location.
NewsAsset	*chef_handler 1.4.0*	*windows 1.44.1*	*newsasset-server*
Description	Distribute and enable Chef Exception and Report handlers	Provides a set of useful Windows-specific primitives.	Custom cookbook to install a NewsAsset application server, its .Net dependencies and a monitoring probe
Oracle DB	*chef_handler 1.4.0*	*windows 1.44.1*	*newsasset-oracle*
Description	Distribute and enable Chef Exception and Report handlers	Provides a set of useful Windows-specific primitives.	Custom cookbook to install a Oracle DB server and NewsAsset application server scheme
File Server	*newsasset-file*		
Description	Custom cookbook to configure CIFS based SMB share		

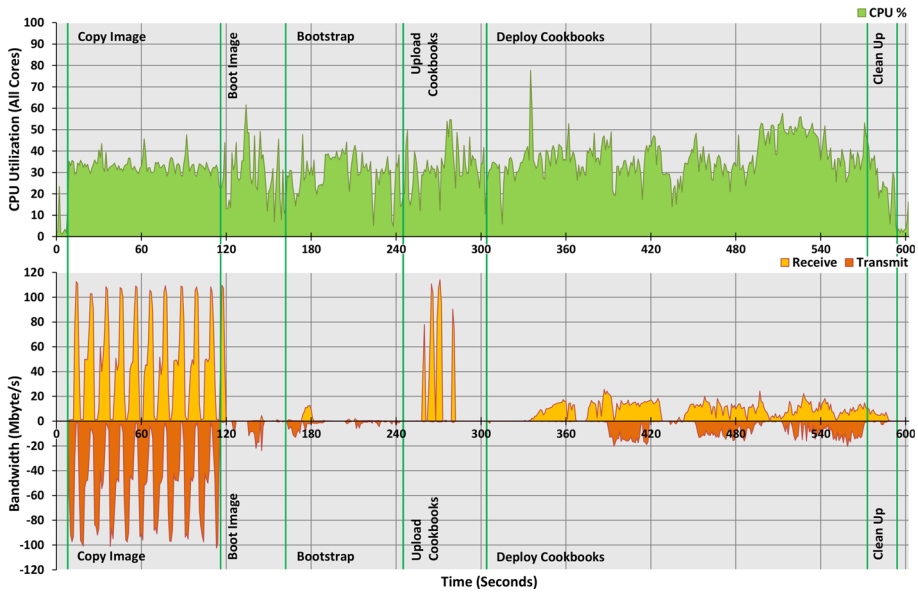

Fig. 10 VMIC resource utility for the generation of a single Windows image

while the deployment of the cookbooks phase is limited by disk I/O. It is recommended that the use of 10 Gbit Ethernet or better and SSD backed storage would dramatically reduce the time to generate images (*recommendation 1*). Furthermore, due to mainly sequential installation processes of the NewsAsset application server, CPU utilization of the VMIC tool is mainly limited to 1 core. Given that the dependency graphs of most software are sequential in nature, we recommend that the VMIC tool be deployed on a machine with the best single thread performance, to reduce runtime further (*recommendation 2*). Finally, the installation of the Chef Client as part of the bootstrapping process could be omitted, if the base images used come with this pre-installed at the expense of additional developer time (*recommendation 3*).

Figure 11 shows the execution time of a range of different NewsAsset components (Windows, Linux, All Components) generated and includes error bars indicating the standard deviation over 10 consecutive iterations of this experiment. It can be seen that the construction of Windows images takes substantially longer than Linux images. This is due to two factors. Firstly, the Windows base images are twice as larger in size (2.6 GByte Windows vs 1.1 GByte Linux) and take three times longer to boot (45 seconds Windows vs 15 Linux). Given that it is difficult to strip down the Windows Operating System to the levels of a minimal Linux installation, it is recommended that developers limit the use of Windows to legacy applications (*recommendation 4*). Finally, from the error bars it can be seen that the distribution of VMIC invocations is tightly packed suggesting consistent performance.

Figure 12 shows the scalability of a range of concurrent users accessing the VMIC to generating the HAProxy Linux component of the NewsAsset Application. It can

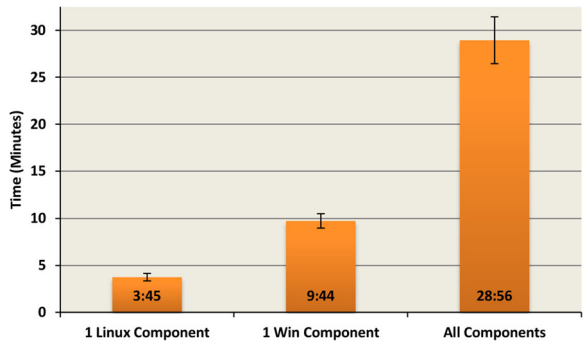

Fig. 11 VMIC Component Performance

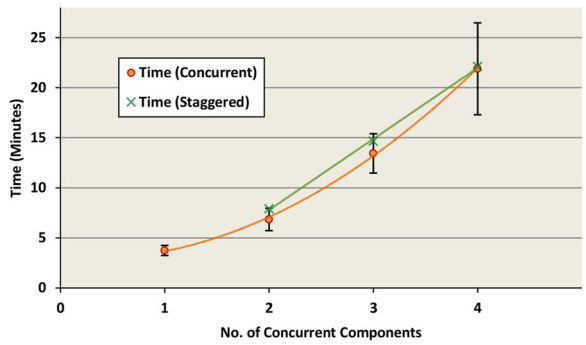

Fig. 12 VMIC Concurrent User Performance

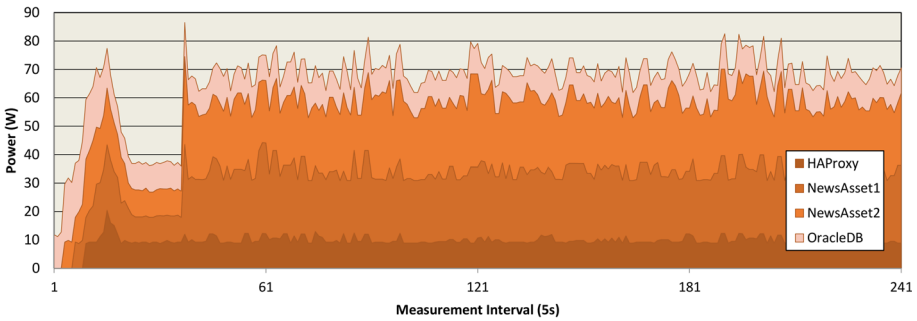

Fig. 13 News Asset Application Server Power Consumption

be seen that as the number of concurrent users increases, the image generation time also increases in a non-linear exponential manner at the same time as variance increases. This is less than ideal if multiple developers wish to share the deployment of a single instance of the VMIC tool. After further investigation of this phenomenon, a second experiment was performed to stagger the arrival rate of concurrent users by 60 seconds. This had the impact of reducing the effect of the Ceph DFS running on 1Gbit connectivity as previously discussed. By spreading its use more evenly, the result of adding 3 minutes of waiting ended in the same runtime for 4 concurrent components generated as without. With this in mind, it is recommended that if the VMIC is to be deployed in a multi-tenant environment, then substantial disk bandwidth and I/O should be available (*recommendation 5*).

In addition to the previous experiments evaluating the performance of the VMIC, Fig. 13 illustrates the benefits of NewsAsset construction with the VMIC and its deployment onto the ASCETiC enabled infrastructure capable of using the embedded monitoring probe. In this experiment the NewsAsset deployment consists of two load balanced application servers driven by a sustained workload. The graph shows the attributed deployment, idle and load power consumption of the NewsAsset VMs during and for a period of a few minutes after deployment. With the VMIC's capabilities to aid in the construction of energy aware cloud applications, the energy efficiency of software can be easily analysed under a variation of loads and deployment configurations (*recommendation 6*).

Conclusion

This paper has highlighted the importance of providing novel methods and tools to support software developers aiming to optimise energy efficiency and minimise the carbon footprint resulting from designing and developing software at the different layers of the cloud stack while maintaining other quality aspects of software to adequate and agreed levels.

We discussed our practical experience during implementation of architectural components relevant to cloud application construction emphasising on automatic image construction, and presented a performance evaluation. The results show that the performance of VM construction is primarily limited by available network and disk I/O. However, for the purpose of a self-adaptive development life-cycle that considers energy awareness the time to generate application components is more than adequate.

Overall, the VMIC is interoperable with cloud providers and technologies that support OVF. It manages software dependency installation and creates cloud provider agnostic VM images in an automated fashion, reducing development effort. Finally, it is shown to be effective through the experimental evaluation of its implementation and is already integrated in a cloud computing toolkit.

Future work on the VMIC will include exploring the integration of third party baseline technologies for the creation of base (master) images such as Vagrant. Integration work will consider the use of containers with the image construction process used in the VMIC such as Docker [41]. We will also consider optimising the energy efficiency of the build process by using DVFS and RAPL policies during the copy-image phase given that it is I/O-bound and can be seen as a possible source of energy saving. This is in addition to research into *intra* and *inter* layer adaptation for the purpose of coordinating the layers of the architecture presented in this paper to further increase cloud application energy efficiency.

Acknowledgments

This work is partly supported by the European Commission under FP7-ICT-2013.1.2 contract 610874 - Adapting Service lifeCycle towards EfficienT Clouds (ASCETiC) project.

Authors' contributions

All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Received: 15 February 2017 Accepted: 18 May 2017
Published online: 23 June 2017

References

1. Pawlish M, Varde AS, Robila SA (2012) Cloud Computing for Environment-friendly Data Centers. In: Proceedings of the Fourth International Workshop on Cloud Data Management, CloudDB '12. ACM, New York. pp 43–48
2. Mastelic T, Oleksiak A, Claussen H, Brandic I, Pierson J-M, Vasilakos AV (2014) Cloud computing: Survey on energy efficiency. ACM Comput Surv 47(2):33:1–33:36
3. Grech N, Georgiou K, Pallister J, Kerrison S, Morse J, Eder K (2015) Static analysis of energy consumption for llvm ir programs. In: Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems. SCOPES '15. ACM, USA. pp 12–21
4. (2016) Packer - Identical Machine Images for Multiple Platforms. https://www.packer.io/
5. (2016) Vagrant - Development Environments Made Easy. https://www.vagrantup.com/
6. Götz S, Wilke C, Cech S, Aßmann U (2011) Runtime variability management for energy-efficient software by contract negotiation. In: Proceedings of the 6th International Workshop on Models@run.time, New Zealand
7. Hilty L, Lohmann W (2011) The Five Most Neglected Issues in "Green IT". CEPIS UPGRADE 12(4):11–15
8. te Brinke S, Malakuti S, Bockisch C, Bergmans L, Aksit M (2013) A design method for modular energy-aware software. In: Shin SY, Maldonado JC (eds). Procedings of the 28th Annual ACM Symposium on Applied Computing (SAC'2013. ACM, New York. pp 1180–1182
9. Grosskop K, Visser J (2013) Identification of Application-level Energy Optimizations. In: Hilty LM (ed). Proceedings of the First International Conference on Information and Communication Technologies for Sustainability (ICT4S'2013), Switzerland
10. Nowak A, Leymann F (2013) Green Business Process Patterns - Part II (Short Paper). In: 6th IEEE International Conference on Service-Oriented Computing and Applications. IEEE, Hawaii. pp 168–173
11. Hönig T, Eibel C, Preikschat WS, Cassens B, Kapitza R (2013) Proactive energy-aware system software design with seep. In: Proceedings of the 2nd Workshop on Energy Aware Software-Engineering and Development. GI Softwaretechnik-Trends. pp 1–2
12. Klingert S, Berl A, Beck M, Serban R, Girolamo M, Giuliani G, Meer H, Salden A (2012) Sustainable Energy Management in Data Centers through Collaboration. In: Energy Efficient Data Centers. volume 7396 of Lecture Notes in Computer Science, Springer Berlin Heidelberg. pp 13–24
13. Mammela O, Majanen M, Basmadjian R, Meer H, Giesler A, Homberg W (2012) Energy-aware job scheduler for high-performance computing. Comput Sci Res Dev 27(4):265–275
14. Armstrong D, Espling D, Tordsson J, Djemame K, Elmroth E (2015) Contextualization: dynamic configuration of virtual machines. J Cloud Comput 4(1):1–15. doi:10.1186/s13677-015-0042-8, http://dx.doi.org/10.1186/s13677-015-0042-8
15. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. Futur Gener Comput Syst 28(5):755–768
16. (2016) CFEngine 3 - Configuration Management Software for Agile System Administrators. http://cfengine.com/
17. Burgess M (2004) Configurable immunity for evolving human computer systems. Sci Comput Program 51(3):197–213
18. (2016) Puppet - IT Automation for System Administrators. http://puppetlabs.com/
19. (2016) Chef - A Systems Integration Framework. http://wiki.opscode.com/display/chef/Home
20. (2015) Open Virtualization Format (OVF) - A standard from the Distributed Management Task Force. http://www.dmtf.org/standards/ovf
21. (2015) OPTIMIS Toolkit. http://optimistoolkit.com
22. Armstrong D, Espling D, Tordsson J, Djemame K, Elmroth E (2013) Runtime Virtual Machine Recontextualization for Clouds. In: Euro-Par 2012: Parallel Processing Workshops. Springer Berlin Heidelberg, Rhodes Islands. pp 567–576
23. Srikantaiah S, Kansal A, Zhao F (2008) Energy aware consolidation for cloud computing. In: Proceedings of the 2008 Conference on Power Aware Computing and Systems, HotPower'08. USENIX Association, Berkeley. pp 10–10
24. Choi K, Soma R, Pedram M (2004) Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In: Proceedings Design, Automation and Test in Europe Conference and Exhibition. IEEE Computer Society, Washington Vol. 1. pp 4–9. doi:10.1109/DATE.2004.1268819
25. Rotem E, Naveh A, Ananthakrishnan A, Weissmann E, Rajwan D (2012) Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. IEEE Micro 32(2):20–27
26. Djemame K, Armstrong D, Kavanagh RE, Ferrer AJ, Perez DG, Antona DR, Deprez J-C, Ponsard C, Ortiz D, Macías M, Guitart J, Lordan F, Ejarque J, Sirvent R, Badia RM, Kammer M, Kao O, Agiatzidou E, Dimakis A, Courcoubetis C, Blasi L (2014) Energy Efficiency Embedded Service Lifecycle: Towards an Energy Efficient Cloud Computing Architecture. In: Proceedings of the Energy Efficient Systems (EES'2014) Workshop. CEUR Workshop Proceedings, Stockholm Vol. 1203. pp 1–6. http://ceur-ws.org/Vol-1203/EES-paper1.pdf
27. Badia RM, Conejero J, Diaz C, Ejarque J, Lezzi D, Lordan F, Ramon-Cortes C, Sirvent R (2015) Comp superscalar, an interoperable programming framework. SoftwareX 3:32–36
28. Lordan F, Ejarque J, Sirvent R, Badia RM (2016) Energy-aware programming model for distributed infrastructures. In: Proceedings of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2016). IEEE, Greece
29. Farokhi S, Jamshidi P, Brandic I, Elmroth E (2015) Self-adaptation challenges for cloud-based applications: a control theoretic perspective. In: Proceedings of the 10th International Workshop on Feedback Computing. ACM, USA
30. Djemame K, Kavanagh R, Armstrong D, Lordan F, Ejarque J, Macias M, Sirvent R, Guitart J, Badia RM (2016) Energy efficiency support through intra-layer cloud stack adaptation. In: Proceedings of the 13th International Conference on Economics of Grids, Clouds, Systems and Services (GECON'2016). Springer, Greece
31. (2016) QEMU - Open Source Machine Emulation and Virtualizer. http://www.qemu.org
32. Bolte M, Sievers M, Birkenheuer G, Niehorster O, Brinkmann A (2010) Non-intrusive Virtualization Management using libvirt. In: 2010 Design, Automation & Test in Europe Conference & Exhibition. Piscataway, USA. pp 574–579
33. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) KVM: The Linux Virtual Machine Monitor. In: Proceedings of the Linux Symposium, Canada Vol. 1. pp 225–230
34. (2015) OpenStack: Open source software for building private and public clouds. http://www.openstack.org/
35. (2015) Zabbix - An Enterprise-class Monitoring Solution. http://www.zabbix.com/
36. (2013) GEMBIRD Deutschland GmbH. EGM-PWM-LAN data sheet. http://gmb.nl/Repository/6736/EGM-PWM-LAN_manual---7f3db9f9-65f1-4508-a986-90915709e544.pdf
37. (2016) ASCETiC. Adapting Service lifeCycle towards EfficienT Clouds. http://www.ascetic.eu/
38. (2016) Newsasset Agency. http://www.newsasset.com/
39. (2016) Oracle Database. https://www.oracle.com/database/index.html
40. (2015) HAProxy - A Reliable, High Performance TCP/HTTP Load Balancer. http://www.haproxy.org/
41. Docker Inc (2017) Docker - Homepage. https://www.docker.com/