CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **00**:1–21
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Distributed Environment for Efficient Virtual Machine Image Management in Federated Cloud Architectures

Dragi Kimovski[1,6*], Attila Csaba Marosi[2], Sandi Gec[3,4], Nishant Saurabh[1], Attila Kertesz[2], Gabor Kecskemeti [5], Vlado Stankovski[3] and Radu Prodan[1]

[1] *University of Innsbruck, Distributed and Parallel Systems Group, Innsbruck, Austria*
[2] *Hungarian Academy of Science, Budapest, Hungary*
[3] *University of Ljubljana, Faculty of Civil and Geodetic Engineering, Ljubljana, Slovenia*
[4] *University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia*
[5] *Liverpool John Moores University, Liverpool, United Kingdom*
[6] *University of Information Science and Technology, Ohrid, Macedonia*

SUMMARY

The use of Virtual Machines (VM) in Cloud computing provides various benefits in the overall software engineering life-cycle. These include efficient elasticity mechanisms resulting in higher resource utilization and lower operational costs. VM as software artifacts are created using provider-specific templates, called VM images (VMI), and are stored in proprietary or public repositories for further use. However, some technology specific choices can limit the inter-operability among various Cloud providers and bundle the VMIs with nonessential or redundant software packages, thus leading to increased storage size, prolonged VMI delivery, stagnant VMI instantiation and ultimately vendor lock-in. Furthermore, the utilisation of centralized repositories, without considering the geographical and logical distribution of the VMIs can aggravate these problems even more, especially in federated Cloud environment. To address the aforementioned challenges, in this paper we present a set of novel functionalities and design approaches for efficient operation of distributed VMI repositories. To this end, we have developed distinctive software modules, specifically tailored for enabling: (i) simplified creation of light weight and highly optimised VMIs tuned for specific application requirements; (ii) VMI repository optimization based on multi-objective approach; and (iii) efficient reasoning mechanism to help optimise complex VMI operations. The introduced novelties have been identified, implemented and integrated as essentials components of the ENTICE environment.

Received . . .

KEY WORDS: Virtual machine images, Distributed VMI repositories, Cloud federation, VMI size optimization, VMI re-distribution

1. INTRODUCTION

Cloud computing technologies are deployed on top of physical hardware in order to provide highly optimised execution environments for Virtual Machines (VM) [1]. The use of VMs (or containers) represents an efficient way to automate the overall software operation life-cycle. For example, elasticity as a main property of Cloud systems usually leads to high resource utilization and thus, lower operational costs [5]. As part of the software engineering process, VMs enable isolated execution of multiple applications within their own environment. VMs are typically created

*Correspondence to: Dragi Kimovski - dragi@dps.uibk.ac.at

using provider-specific templates, called VM images (VMI) and are stored in proprietary or public repositories for further use. However, some technology specific choices, such as virtualization environment and VMI format, can limit the inter-operability among various Cloud providers and bundle the VMIs with non-essential or redundant software packages, thus leading to increased storage size, prolonged VMI delivery, stagnant VMI instantiation and ultimately vendor lock-in. Furthermore, the utilisation of centralized repositories, without considering the geographical and logical distribution of the VMIs can aggravate these problems even more, especially in federated Cloud environment.

Regrettably, current state-of-the-art solutions do not provide any substantial methods and tools for optimized operation of distributed VMI repositories, thus limiting the adoption of federated Cloud environments [3]. Multiple crucial constrains that largely influence the usability of federated Cloud infrastructures, in the sense of VMI repository operation, have been identified as follows: (i) manual, time consuming and error-prone VMI creation with proprietary templates, (ii) non-divisible VMIs, containing unnecessary and redundant software packages, (iii) centralized and unoptimised VMI repositories, and (iv) lack of information to support the process of VMI and repository optimization [4].

To address the aforementioned constraints, in this paper we present a set of novel functionalities and design approaches for efficient operation of distributed VMI repositories in federated Cloud environment. To this end, we have developed distinctive software modules, specifically tailored for enabling: (i) simplified creation of light weight and highly optimised VMIs tuned for specific application requirements; (ii) VMI repository optimization based on multi-objective approach; and (iii) efficient reasoning mechanism for streamline support of complex VMI operations.

The introduced novelties have been identified, implemented and integrated as essentials components of the ENTICE environment [2] [†]. The ENTICE environment, currently under development, is an advance architecture capable of receiving unmodified and fully functional VMIs from its users, and openly adapt and optimise them for specific Cloud infrastructures with respect to various criteria, such as, optimal size, configuration, and geographical distribution. ENTICE employs novel technologies to enable faster VMI loading and delivery, and optimized application execution with improved QoS. ENTICE gradually stor-es information about the VMIs and fragments in a knowledge base that is used for interoperability, integration, reasoning and optimisation purposes. VMI management is supported by ENTICE at an abstract level, independent of the middleware technology supported by the Cloud infrastructures within the federation. To further shield the users from the complexity of underlying VMI repository technologies, ENTICE focuses on providing the flexibility for tailoring the VM images to specific Cloud infrastructures. The ENTICE repository includes, among other features, techniques to optimise the size of VMIs while maintaining their functionality, automatically share similar parts of multiple images, such as identical OS kernels and libraries, among various repositories located in different administrative domains, and optimally distributed them in response to application and data-center requirements. In a broad sense, the ENTICE environment, aims to provide a universal backbone architecture for efficient support of VMI management operations, regardless of the administrative boundaries between multiple distributed repositories.

The remainder of this paper is structured as follows. In Section 2 we provide relevant related work concerning the state-of-the-are concepts in VMI repositories management. Afterwards, in Section 3 we present generalized use case scenario for the ENTICE environment and it's essential modules. In relation to the use cases, we also provide analysis and identification of the essential requirements. In Section 4, we present the system architecture of the ENTICE environment and identify the essential functions required for efficient VMI management.

Section 5 encloses the VMI synthesis and analysis concepts and related approaches. Later, in Section 6 we present the Multi-objective optimization framework with replication support. Section 7 present novel concepts pertaining to the knowledge base and reasoning mechanism. In Section 8

[†] http://www.entice-project.eu/

DOI: 10.1002/cpe

we present experimental evaluation of the developed functional modules by considering multiple different scenarios. Lastly, section 9 summarizes the main achievements of this work, their overall impact and future work directions.

2. RELATED WORK

Due to the highly dynamic demand for storing and managing VMI files in distributed Cloud environment, the traditional VMI management systems are not able to provide support for complex VMI operations, such as federation wide VMI redistribution or VMI size optimization and similarity based fragmentation. To begin with, we present few existing VMI management systems, such as VMware [22] repository system, which at its disposal allows upload and download of images by authorized users. In addition, the VMware system offers a weak management system for locally stored VMIs within a single repository and provides efficient tool for searching specific VMIs in accordance with application requirements. Another VMI specific repository management environment, offered by Science Clouds [23], allows download of existing stored images, but evades the upload or indexing functionality of any user specific VMIs. FutureGrid [9], an experimental system for HPC and Cloud based applications, provisions another image repository targeted at federated storage systems, which empowers the users to upload, download and update VMI functionality with limited meta-data informations through REST interface.

Apart from above mentioned systems, there are few VMI repositories namely, *VMRC*[10], and Amazon Image Service [24], which come closest to solving significant part of issues pertaining to the efficient VMI management. On one hand, VMRC offers indexing of images via *Catalog* functionality, while Amazon allows publish/share of most common VMIs with respect to appropriate authorization. However, Amazon comprise of proprietary image repository, tying down the users to a specific cloud hypervisor environment and VM template. VMRC as well does not contribute towards solving the interoperability issue, and instead offers a unique VM matchmaking service for sharing of images among the local users. Moreover, the locally centralized architectural model, of the mentioned VMI repository systems, does not provide scalable image distribution and hence induces delays in the VM provisioning. To this extent, none of the mentioned production systems contributes towards bridging the gap that limits the deployment of federated IaaS cloud model.

Other than the popular production systems, there has been limited individual research work focussing towards specific independent goals aiming to solve the VMI storage and distribution issues. Kaveh et al. [11] optimizes the VMI storage cost, based on a reduction of the image size to the bare minimum required for executing a given application. The effective method of pruning VMIs, in this case, improved the startup time of virtual machine instances by up to three times. On the other hand, with regards to VMI distribution, Wu et al. [12] provides theoretical work based on isolated and cross image distribution mechanism optimizing the average distribution time of all VMIs in the repository. The isolated model emphasizes on creating swarm structures where data exchange corresponding to instantiation of VMI is confined within individual swarms, while in cross model distribution, swarms exchange VMI common parts. Schmidt et al. [15] extends this discussion by including other structural mechanisms such as unicast, binary tree, bit torrent and multicast distribution of VMIs and concludes that the P2P based bit torrent mechanism is most efficient compared to all others, specifically in case of remote VMI distribution.

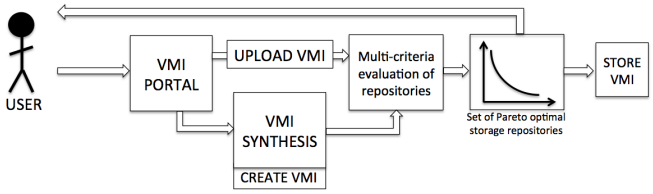
Although, there has been significant advancements in research with regards to the VMI related operations, some important aspects have been left open. Furthermore, there is no VMI management environment capable of encapsulating the required complex VMI operations under a single perspective. Hence, there is a need of a generic VMI repository environment, which gives transparency to its users with regards to automated contextualization, reduction and fragmentation of VMIs and optimizes the storage distribution with the support of a semantic reasoning based management.

DOI: 10.1002/cpe

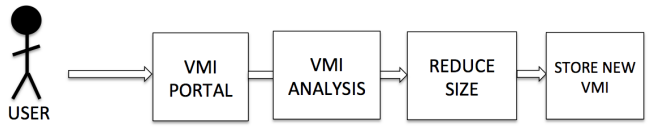
3. USE-CASE SCENARIOS AND REQUIREMENTS ANALYSIS

To tackle the issues that limit the possibilities for Cloud federation and identify the most relevant requirements we present a broad use-case scenario of the proposed environment. Furthermore, based on these requirements, in the following section we identify the imminent functionalities of the ENTICE architecture.

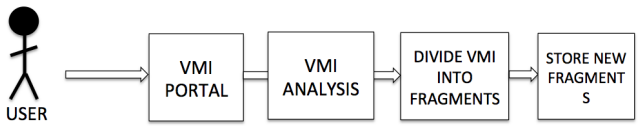
The use-case of the ENTICE environment typically commence with the user issuing request for storing unmodified and functionally complete VMI, through an easy to use user interface, enclosing various VM management operations. Subsequently, the environment transparently evaluates multiple different storage repositories where the VMI can be initially stored. The evaluation is conducted based on multiple conflicting criteria, such as cost for storing or delivery time. This process results in a set of optimal trade-off solutions, i.e. initial storage repositories, from which the user should choose only one. Alternatively, instead of uploading complete VMI, the user can choose to create new highly optimized VMI by utilizing generic pre-specified recipes and novel synthesis techniques (1a).



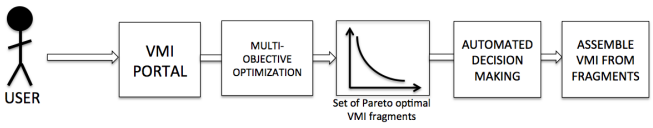
(a) VMI upload



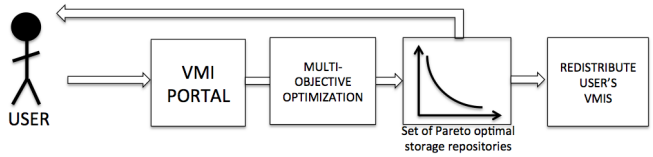
(b) VMI size reduction



(c) VMI fragmentation



(d) VMI assembly



(e) VMIs redistribution

Figure 1. Use-case scenarios for the ENTICE environment

DOI: 10.1002/cpe

In general, user specific images are large sized and comprise various redundant and unessential packages, thus incurring high storage cost and increased instantiation overheads. Henceforth, the user may require for the newly uploaded VMI to be optimized in size. To this end the ENTICE environment is envisioned to be capable of removing redundant packages and files from the VMIs by introducing efficient analysis techniques (1b). Furthermore, the VMI analysis should enable discovery of identical portions in apparently unrelated VM images, such as identical OS kernel or library, coming from different users. Based on the similarity information, the environment may allow spiting of VMIs into multiple fragments for storing the frequently shared components only once (1c). Later, if the user requires for a specific image to be deployed in a computational Cloud, the ENTICE environment will enable utilization of multi-objective optimization techniques for fast assembly of the previously identified fragments in a stand alone VMI and convert it in the appropriate Cloud template (1d). In addition, to reduce the storage costs or to increase the deployment performance, the user can also require more optimal distribution of it's own images among the register ENTICE repositories (1e). To this end, the ENTICE environment will transparently redistribute the VMIs based on a novel multi-objective optimization techniques. The process itself will be conducted online during application execution, or offline while the VMIs are not being utilized or optimized. The VMI redistribution is proceeded by a Service Level Agreement (SLA) capable of properly defining multiple optimal trade-off solutions. The composite trade-off SLA is then provided to the user, which acts as a decision maker for the optimization process.

Based on the general use-case scenario, we have identified the requirements for the ENTICE environment. Hence, we divide the ENTICE design requirements in the following broad categories:

- *VMI synthesis, analysis and fragmentation requirements* enclosing the specification of the recipe based VMI creation process, VMI size optimization and reduction, and VMI fragmentation and assembly.
- *repository optimization and VMI distribution requirements* covering the needs for more efficient initial distribution of newly uploaded VMIs, redistribution of user specific VMIs, online VMI assembly and VMI interoperability support among various Cloud providers.
- *knowledge base and reasoning requirements* encapsulating the proper storing of relevant usage data for efficient functioning VMI analysis and fragmentation modules. Furthermore, these requirements are extended to cover the support for VMI redistribution and constrain based multi-objective optimization with Service Level Agreement (SLA) support.

4. ENTICE FUNCTIONALITIES AND SYSTEM ARCHITECTURE DESIGN

To satisfy the design requirements, the ENTICE environment should transparently support various complex functionalities, without requiring deep involvement of the users. By systematically analysing the use-case requirements the functionalities for the ENTICE environment can be identified and classified into three distinctive categories:

- *VM Image synthesis, analysis and fragmentation functionalities, including*: (i) recipe based VMI synthesis, (ii) VMI size optimization by removing parts of the operating system which are not required and have not been defined in the functional description, (iii) division of the VMI in multiple fragments with respect to the required functionalities, thus removing duplicated software packets across multiple images and (iv) assembly of fragments into complete VMI during the deployment stage.
- *Multi-objective VMI distribution optimization with replication functionalities, including*: (i) applying Multi-objective optimization for distribution of the VMI and fragments across distributed storage sites, (ii) optimized extraction of fragments, based on functionality, for assembly of the VMI during the deployment stage, (iii) movement of VM image from one storage location to another, and (iv) location tracking of each VMI to enhance storage and distribution.

DOI: 10.1002/cpe

- *Knowledge based VMI portal and reasoning functionalities, including*: (i) an easy interface for the users to access available VMI and to search fragment,(ii)VMI upload and update functionality for the streamlined provision of user's images, (iii) VMI management and graphical analysis tool for the Pareto-SLA module, which allow for the users to choose SLA's from multiple optimal metrics provided by the Multi-objective optimization framework,(iv) functionality to sort VMI with respect to user defined metrics, and (v) easy interface to User profile management.

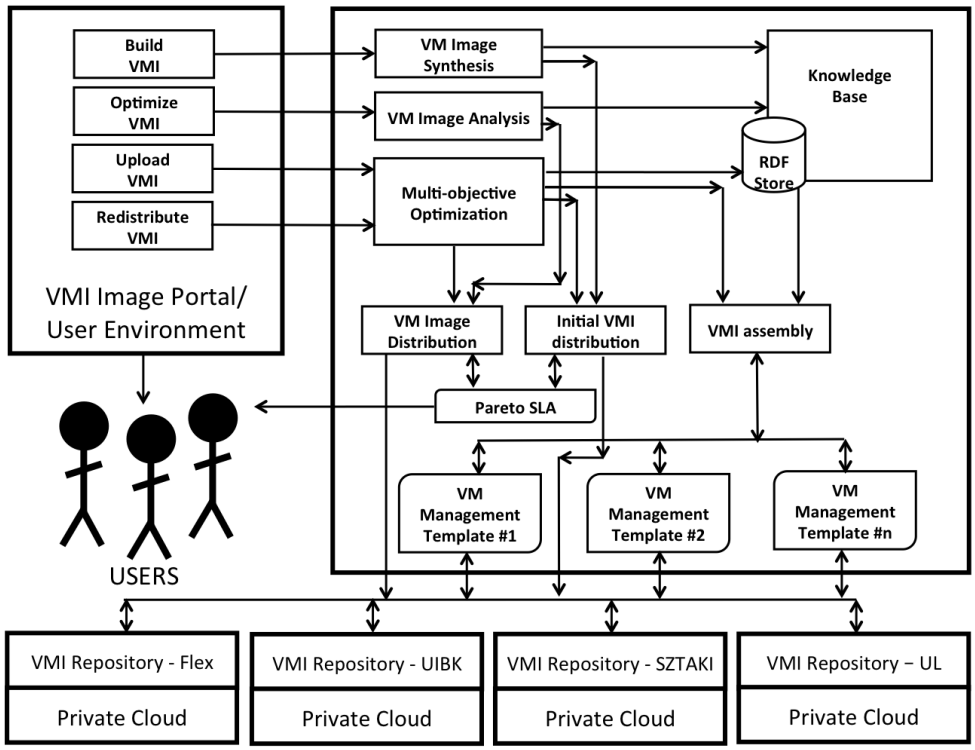


Figure 2. Functionalities supported by the ENTICE environment

The above-listed functionalities were the main guiding point in the design and development process of the researched environment. From architectural point of view, all these functions make ENTICE a perplexing and highly complex system. To streamline the development process of such system, a distributed and highly decentralized approach was taken. The architecture of the research environment is modular in nature and encapsulates variety of different components which interact by exchanging structured information on the available services. Each component in the system provides specific functions that either supports set of complex VMI operations or enables proper functioning of other components. The communication between the software components is established by utilizing RESTful protocol, while the transfer of VMIs and related datasets is based upon secured HTTP protocol. The current state of the ENTICE architecture, together with the corresponding software components has been depicted on Figure 2 and is composed of a:

- Client-side VMI image portal, through which the users can interact with the environment and initiate various complex VMI operations, such as VMI synthesis, VMI upload, VMI size optimization or VMI redistribution. Furthermore, the VMI portal, provides output information to the users and efficient tools for decision making on the SLA contracts.

DOI: 10.1002/cpe

- Distributed services enabling the complex VMI operations. The provided services have been distributed across multiple geographical regions. For example, the multi-objective optimization framework, together with initial VMI distribution and VMI redistribution modules, have been deployed in the premises of the University of Innsbruck in Austria, while the VMI synthesis and analysis tools have been integrated within the production class public Cloud of the Hungarian Academy of Science in Budapest. All these services are supported by a knowledge base and advance reasoning mechanism located at the University of Ljubljana in Slovenia.
- Lastly, the ENTICE architecture encloses the registered VMI repositories, which again have been distributed on various location through Europe.

In what follows an extensive overview of the distributed services, enabling the complex VMI operations, is provided. More concretely, the research and development approaches for each separate module of the ENTICE environment has been presented, followed by performance and behaviour evaluation.

5. VMI SYNTHESIS AND ANALYSIS

As introduced in Figure 1 of Section 3 VMI synthesis and analysis is also an important case in ENTICE. By defining these processes our goal was to identify a simple to follow methodology usable to fragment (i.e. split up) a monolithic application alongside its sub-service boundaries – that is what we call as fragmentation. These sub-services can act as small micro-services that later can be composed to other services without the need of the entire monolithic application. To achieve this, we use image synthesis and image analysis methods, which both have pivotal roles within the architecture of the ENTICE project. We previously [17] presented our approach at a high-level, in this section we further elaborate and describe these mechanisms, and present experimental evaluation in Section 8.1.

The ENTICE environment offers virtual machine management templates (VMMT) to be stored in the repositories of the connected cloud systems. These VMMTs allow the fragmented images to be reconstructed at runtime. For optimal VM instantiation performance, the templates are formulated as stand-alone VM/Container images to access and build fragments from the distributed repository. After a VM is instantiated using a VMMT, it will ensure fragments (needed for a particular functionality specified by the user) are placed and enabled for use in the instantiated VM. VMMTs even allow customisation of files/directories for specific VMs to serve the needs of various stakeholders.

The VMI synthesis enables users to build new images with several approaches. First, it allows the use of generic user provided images or software recipes to act as the foundation before specialising them into microservices. Second, VMI synthesis cooperates with the ENTICE image portal to identify the functional requirements a newly created image must meet (per microservice basis, thus resulting in a new image for every functional requirement specified). Finally, our synthesis tool modifies the generic (original) images either directly (by altering the image file(s)) or indirectly (through creating alternative recipes that lead to more compact images). These alterations aim at removing contents from the original images: the alterations lead the generic images towards their single purpose (namely, the functional requirements listed against the image in the portal). For optimised images, the VMI synthesis offers image maintenance operations (e.g., allowing software updates to be done on the original image and transforming those updates to the optimised image).

Alongside synthesis, ENTICE also delivers VMI analysis allowing the discovery of equivalent pieces in apparently non-related VM images possibly originating from different users and communities. Analysis operates independently of the cloud provider where the image is stored. Equivalence information then stored in the ENTICE knowledge base for later use.

ENTICE also allows dividing VM images into smaller fragments allowing the storage of some common image components only once (e.g., same Linux distribution used by two unrelated images).

DOI: 10.1002/cpe

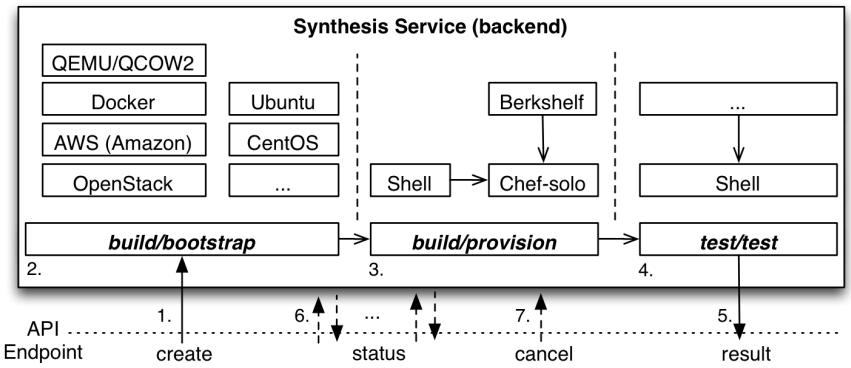


Figure 3. Process of recipe based image synthesis

Fragmenting fosters the VM image distribution and enables the optimization of overall storage usage in the repository.

5.1. Recipe based image synthesis

This subsection is focused towards presenting the ENTICE functionalities enclosing the creation of new highly optimized VMIs by using user's specific functional recipes. On this level, the recipes are expected to guide the creation of the user's original monolithic service on a generic way.

The recipe based image synthesis process of ENTICE can be seen in Figure 3. It depicts 7 steps starting by creating an image, and ending with an optional cancel request. ENTICE provides APIs in a REST interface to use the services covered by these steps. There is also a backend part of this Synthesis service that uses other subcomponents to create the requested images. The images that can be managed in these processes may be of normal virtual machines (e.g. VMIs) or containers. The contents can also vary from microservices to complex ones. As they suggests, microservices in containers have smaller footprints, therefore they are easier to optimize. As shown in Figure 3 the API enables the following processes:

- submit build requests (see step no. 1 in Figure 3);
- retrieve results of finished builds (see step no. 5 in Figure 3);
- query the status of ongoing and finished builds (see step no. 6 in Figure 3);
- and cancel in-progress builds (see step no. 7 in Figure 3).

The image creation process at the backend consists of two parts. The first one is the building phase, while the second is the testing phase. First let's detail the building phase. It can be initiated with the create API call (step no. 1 in Figure 3) by specifying the parametrized build target, the service description for provisioning, and the tests for the testing phase.

The first part of the building phase is the bootstrapping step (no. 2). It is responsible to make a base image (in case of VMI's) or a container available for the provision step. It is possible to create them in the following ways:

- start from an empty disk image and build locally (via QEMU/QCOW2);
- target a container (e.g., Docker);
- build on an existing image from a repository via a supported cloud (e.g., OpenStack). In this case no bootstrapping is performed.

The second part of the building phase is provisioning. It installs the requested microservice using the specified description. This can be done in two ways. First is the Shell provisioner (via Packer). Here a custom shell script contains all steps to be executed. Another possible option is to use some provisioning tool like Ansible or Chef(-solo). In case of Chef-solo, cookbooks must be provided,

DOI: 10.1002/cpe

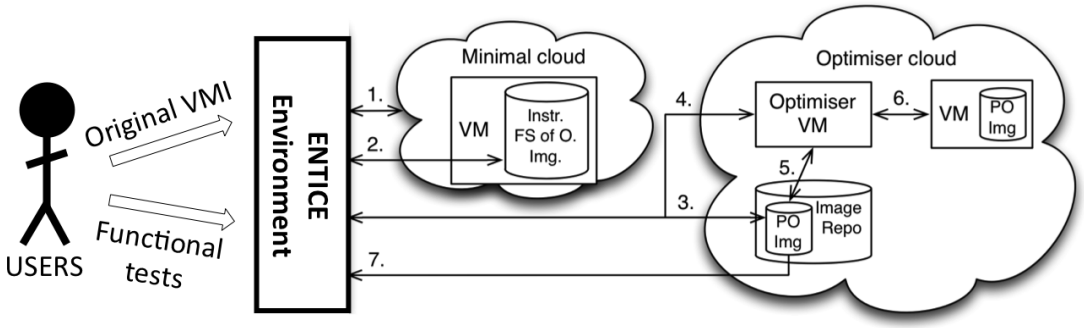


Figure 4. Steps to transform an image to host only the intended microservice

e.g., first fetched via Berkshelf and added to the build description, but custom ones can be used as well. These provisioner may be used together when needed, e.g., performing housekeeping via Shell and provision the microservice components using Chef.

During the testing phase, the image is duplicated, and the supplied test script is executed in the copied image. The testing methods can be of any type, only the exit status differentiates: a non-zero represents an error. The script can deploy any components from the repositories of the Linux distribution and a user supplied custom zip file may contain additional components as well, but all other external access is prohibited. The testing phase allows flexible methods since different services might require vastly different approaches and tools for testing. The image where the tests were executed is discarded after, while the original can be download via the API. Currently there is no option to link the image to another location or repository, this feature will be considered for future work. Our current implementation uses Packer [19], but our initial experiments were carried out using ImageFactory [18].

5.2. Targeted Size Optimisation

Next, we detail the VMI size optimization functionality, presented in Figure 4. This developed module can be executed after the recipe based synthesis finished building a monolithic VM or container image. These monolithic images are referred here as the original images (denoted as *O. Img.*) of an application. Usually these monolithic images implement a combination of several functionalities, therefore the user can use the ENTICE environment to transform the original image to an optimized one, which keeps only the required functionality (resulting in a microservice). Before the microservice can be extracted from the original image, the image creator should prepare a functionality test for the required microservice (in the form of independent, self-contained shell scripts). These tests must utilise all features of the required microservice. In general, unit tests of the original application could be a good start for the functionality test. Although these tests need manual preparation, they are sufficient to be used in proof of concept scenarios. In the future we will develop techniques to enable semi-automatic functionality test generation.

These transformations used for VMI size optimization are done in 7 steps as shown in Figure 4:

- Step 1. Instantiate the original image in a minimal cloud infrastructure.
- Step 2. Execute user tests and collect accessed files.
- Step 3. Create a partially optimised image file.
- Step 4. Deploy the Optimizer VM.
- Step 5. Deploy the partially optimized image, and remove further unnecessary parts.
- Step 6. Run functionality tests for the optimized image.
- Step 7. Upload the optimized image to the ENTICE environment.

Next we detail these steps. Once the test is uploaded to the ENTICE portal, the pre-evaluation phase starts. In step 1, the original image is instantiated in a minimal cloud infrastructure maintained by the ENTICE environment. ENTICE's minimal cloud runs the new VM or container and

DOI: 10.1002/cpe

instruments its file read operations (i.e., *Instr. FS* in the figure). Then the functionality test is ran on the recently created VM/container. While this test runs, ENTICE collects the files accessed from the original image in step 2. If this initial test fails, the collected data is discarded and the user is notified about the incorrect test result for the original image (suggesting that the actual test requires more functionality than the original image offers). Upon success, the collected list of files are stored marked in the so-called *restricted list*. We assume that files not on the restricted list are not relevant for the microservice identified by the functionality test.

In step 3, which is part of the image optimisation phase, the system creates a partially optimised image (PO image) that contains only the listed files (i.e., non-referenced files are erased). In step 4, an Optimizer VM is deployed and contextualized so it should use the PO image, and perform the optimisation procedure. In step 5, The Optimizer VM starts by altering the PO image (i.e., remove some further contents from it), before the test script is run. The removed parts of the image are expected not to be relevant for the microservice's intended functionality, instead they are assumed to belong to the background activities of the original image (e.g., startup functionalities and periodic tasks unrelated to the mircoservice's functionality). Since in this paper we present and describe the methodology (inner workings) of the ENTICE environment, we do not introduce specific selection techniques to applied on the parts of the PO image for further optimisations.

Once the PO image is altered by the removal of further files, the Optimiser VM uploads the new image to the cloud in step 6, and tests the image by instantiating it and evaluating its functionality via the user-provided shell script. Success in the evaluation leads the optimizer to use the altered image as the new PO image. If the evaluation fails, other candidate files are selected for removal from the previously examined PO image. The steps (3 to 6) in the optimisation phase are repeated until the user-defined cost limits are not reached, or till no more image components can be selected. By the end of this step the final PO image will be ready (containing only the intended functionality of the microservice), which is handed back to the ENTICE environment in step 7.

It is expected that the original image creator modifies the service interface for the optimized image after the optimization process, since the microservice offers a subset of the features or functionalities of the original image. Thus, it is reasonable to reduce the interface, too. This is a manual operation, and the optimisation phase should be rerun with the altered interfaces to validate its usability. As the optimization framework for VMI optimizer reuses the past file selection results (i.e., the ones successfully removed in step 5), this second optimisation phase will be much faster.

6. MULTI-OBJECTIVE VMI OPTIMIZATION FRAMEWORK WITH REPLICATION SUPPORT

Cloud providers currently store VMIs in a proprietary centralized repositories without considering application characteristics and their runtime requirements, thus causing high delivery overheads. The ENTICE environment evolves from the typical storage systems, and instead focuses on providing means for transparent distribution of VMIs in distributed storage repositories by considering application characteristics and usage patterns. To achieve this rather ambitious goals, an extensive research was conducted on a generic multi-objective optimisation framework for VMI redistribution, targeting a number of important objectives such as performance related, operational cost and storage size, applied to multiple particular problems. The researched framework utilizes unified multi-objective optimization module, which has been branched in three distinctive sub-modules tailored for specific optimization tasks [8].

To support the process of optimal VMI distribution it is essential at the upload stage to consider multiple performance and financial objectives. To this end, the process of initial VMI upload initiates the optimization framework to search for most optimal upload repository according to user specific requirements. The framework utilizes concepts from the field of multi-criteria evaluation to determinate the possible repository sites where the VMIs or associated data sets can be initially stored.

DOI: 10.1002/cpe

To reduce the VMI delivery times for complex requests, the optimization framework is capable of performing automated optimization of the VMI distribution across multiple repositories in the background. The concept of background or offline VMI Redistribution is built upon specifically designed optimization module, which considers VMI usage patterns to re-distribute the images in accordance with multiple conflicting objectives. The optimization algorithm provides multiple trade-off solutions, where each solution represents a possible mapping between the stored images and available repository sites.

Furthermore, the use of replication on the highly demanded VMIs during the offline optimization has been investigated. The size of the data that needs to be accessed on the current VMI repositories is on the order of Terabytes and it is soon expected to reach Petabytes. Therefore, ensuring efficient access to widely distributed VMI and associated data-sets is a serious challenge that needs to be addressed. Creating replicas to a suitable site by implementing VMI replication strategy can increase the system performance, shorten the data access time and increase the system availability.

To properly distribute the VMIs in federated environment the optimization framework is dependent on a careful analysis of the repositories usage patterns. To this aim a specific module is required to store and process information on the VMI activities and to provide the collected data in a proper format. To this end the ENTICE knowledge base has been utilized and interconnected with the optimization framework. The framework itself, has been designed to acquire input data from the knowledge base, and also to return the output results there. Furthermore, the framework provides a SOAP based API, through which the decision maker can acquire the set of optimal trade-off solutions in a visualized manner, thus reducing the complexity of the VMI storage management process.

In what follows a comprehensive overview of the VMI redistribution module with replication support has been provided.

6.1. VMI redistribution with replication support

The optimization problem of offline VMI redistribution with replication consist of a finite, although very large number of possible alternatives. The identification of the optimal set of distribution possibilities requires proper modelling of the conflicting objectives. More concretely, the optimization process is conducted by utilizing three objectives: cost for storing and transferring of VMIs, system reliability and performance objective enclosing the VMI delivery time. The modelling of the objectives has been performed by analysing the VMIs usage patterns, thus resulting in optimized image distribution across multiple distributed repositories.

6.1.1. Cost objective To model the cost objective the framework focuses on the notion of the financial resources which are required to store particular VMI in a given repository site C_{st} and the burden for transferring the data from the initial to the optimal repository in the federation C_{trnew}. For each VM image the cost objective can be calculated by summing the transfer and storage costs in accordance with the repository price list.

6.1.2. Performance objective On the other hand, the modelling of the performance objective has been conducted by approximating the average throughput between two know points within the Cloud federation. Utilizing the raw theoretical throughput of the interconnecting structure to describe the factual communication performance is not a suitable approach. In practice it is usually difficult to predict the actual route the packets may take through the interconnection network to reach the destination, thus increasing the performance uncertainty. Furthermore, the load on the intermediate communication channels can deviate, rendering the theoretical communication performance unsuitable for the researched purpose.

To this end, the optimization framework leverage the VMIs usage patterns, stored in the knowledge base, to perform close approximation on the actual throughput between any pair of nodes in the federation. Therefore, if there is a sufficient historical data on the previous transfers among the repository sites and the computing instances, virtual links between the above mentioned

DOI: 10.1002/cpe

entities can be abstracted. Alongside, the average bandwidth of the modelled virtual links can be estimated. The estimation of the virtual links bandwidth B_{rc_i} is conducted based on the previously recorded transfer events by the monitoring module. The monitoring data is used to determinate if there have been any previous VMI or data transfers between a given cloud and a repository. The average value of the throughput for every single transfer is then taken as a performance metric for the virtual link.

The introduction of virtual links, allows for an undirected fully connected weighted graph to be modelled between a particular storage repository and all registered computational Clouds. In this graph, the vertices correspond to either a single repository site or a computational Cloud instance and the edges are represented by the "virtual" links. The weights of the edges correspond to the estimated average bandwidth B_{rc_i} on the corresponding "virtual" links.

To properly model the performance objective for storing a VMI in a given storage repository, we introduce so-called performance adjusting function, which considers the total number of downloads of single VMI from particular repository to all Clouds D_{tv} and the number of downloads to exact neighbouring Cloud D_i. The ratio of those two values is then multiplied with the estimated bandwidth of the corresponding "virtual" link to adjust for the delivery frequency relevance.

Subsequently, this process is then repeated n times, until the adjusted performance has been calculated for every single virtual link emerging from a given repository. Lastly, the average value from this process is weighted as a performance objective for storing a single VMI in an exact repository. Therefore, the performance objective has been modelled based on the following equation:

$$P = \sum_{i=1}^{n} B_{rc_i}(\frac{D_i}{D_{tv}}) \tag{1}$$

6.1.3. VMI availability objective Introducing the concept of replication for VMI redistribution encloses multiple different aspects, as it can affect both, the system performance and the VMI availability. Consecutively, we propose a specific replication strategy focused on increased availability and reduced access time of the most frequently used VMIs.

To determinate which VMI has to be replicated we primarily consider the download frequency. First the optimization framework calculates the median download frequency $D_{f_{med}}$ for all VMIs stored within the ENTICE environment. Afterwards the difference between the median and maximal value for the download frequency is determinate. This value is then multiplied by the replication coefficient k, where $0 < k < 1$. Therefore, if $k = 0$ none of the VMI images will be replicated, otherwise if $k = 1$ then half of the stored VMIs will be replicated by factor of one. The value of k has direct influence on the number of VMIs that will forego the replication procedure, thus directly influencing not only the availability of the VMIs, but also the cost and performance objectives.

Due to the aforementioned reasons, we have decided to isolate the system availability value as a separate objective and to consider the influence of the replication procedure in the evaluation of the performance and cost objectives. In the cases when specific VMI needs to be replicated, the replica is treated same as the original image in context of the cost and performance objectives and the objective values are calculated in accordance with methods presented in the previous Section.

The VMIs availability VA is then evaluated as a separate objective, by taking into account both the original and replicated VMIs. In the general case the availability objective is calculated by using the equation below:

$$VA = \prod_{i=1}^{o_{nr}} R_i \times \prod_{j=1}^{r}(1 - \prod_{k=1}^{N_r}(1 - R_i)) \tag{2}$$

, where o_{nr} is the number of original VMI that will not undergo replication, r is the number of VMI that will be replicated and N_r is the number of replicas that are going to be created for a single VMI.

DOI: 10.1002/cpe

6.2. *Search algorithm and population modelling*

The VMI redistribution module utilized NSGA-II search algorithm to perform the optimization process [7]. As with any other population based genetic algorithm, the individuals are used to represent and evaluate multiple possible solutions to the problem.

For the purpose of VMI redistribution we have represented each individual as a vector with a size equal to the number of images that will be redistributed and then replicated. The values stored in each of the vector fields corresponds to a storage repository where a particular VMI or replica can be stored. Therefore, every vector contains unique mapping between the redistributed VMIs and replicas to the available storage sites in the federated repositories.

Based on this principle the initial population for the NSGA-II algorithm is created. The values in the vector fields are then randomly generated with bounds ranging from one to the number of available storage sites. Afterwords, for each field i of the vector the performance P_i and cost C_i are calculated in accordance with the procedures explained in the sub-section above. Those values are then grouped together and the average value is selected as the overall fitness of the given individual in accordance with the cost and performance objectives. The VMI availability objective is calculated for the overall individual, as described in Section 6.1.3. The vector structure for an example individual containing seven original VMIs and three replicas has been presented on Figure 5.

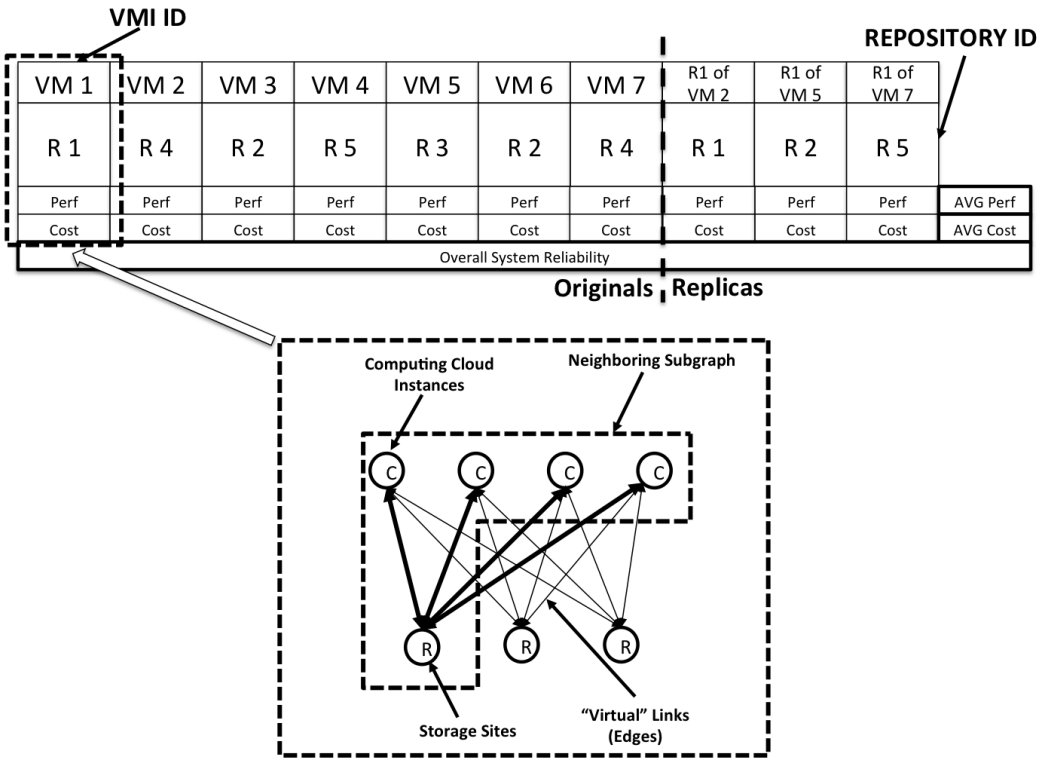


Figure 5. Individual vector for the VMI re-distribution algorithm

When the evaluation process of the initial population has been completed, the proper mutation and crossover operators are applied to create children population of the same size. Next, the children populations is evaluated and grouped together with the parents population. The grouped population is then sorted according to dominance. The individuals for the next generation are selected from the grouped population. This process is afterwards repeated until the maximal number of generations has been reached.

DOI: 10.1002/cpe

The solutions which have been acquired after the last iteration are finally sorted based on the dominance. The optimal trade non-dominated solutions are then presented to the administrative entity of the federation, which acts as a DM, and should select the most appropriate VMI distribution solution based on the pre-defined decision making policy.

7. KNOWLEDGE BASE REASONING AND DECISION MAKING

An important hypothesis of this work is that knowledge management methods can help address a variety of problems stemming from the wide range of needed functional and non-functional properties of software artefacts (VMs) and the complexity of the federated repository in which they are stored and managed. Thus, the choice to use knowledge base for storage and complex querying and even inference of new knowledge from the data, is the most suitable in comparison to relational and other types of storage technologies.

The implementation role of the knowledge base in the overall system consist of three main aspects: (1) design of an ontology describing the entities used in the ENTICE environment, which included models of functional and non-functional properties of VMIs, model of the individual storage repositories and so on, (2) development of a knowledge base web based service that supports CRUD (create, read, update and delete) operations to manipulate with RDF based instance data and (3) implementation of reasoning mechanisms to automate the process of using the knowledge. Moreover, the knowledge management methods are reflected in the VM image portal and support the Pareto-based MO framework.

This section is logically split into two parts; the first part focuses on high-level concepts, which are technology prone as-much-as-possible, relevant for ENTICE, and include ontology with reasoning in a form of objects, properties and relations between them as well as an architectural design; the second part describes implementation details, including the selection of the technologies and some issues and/or challenges related to them.

7.1. Conceptual modelling

Initially, it was necessary to build an ontology for ENTICE, which is a conceptual model of all the entities, their attributes and relationships in the complex ENTICE environment. The ontology design basis on identified use cases, their data flow and data definition of the service behavior. Instance data for the Knowledge Base can be generated and retrieved from different sources, for example, from the VM images portal, from the individual Cloud repositories and their Service-Level Agreements (SLAs) and similar. All these entities had to be modeled with the ENTICE ontology. The ontology is designed to support also temporary lifecycle states and statistics. Some of these data has to be stored persistently for future decision making, reasoning and some algorithmic calculations.

ENTICE gradually stores information related to all the concepts in a Knowledge Base that will be used for interoperability, integration, reasoning and optimisation purposes. These concepts could be Cloud resources (e.g. ENTICE components and their environmental functionalities, storage resources, VMIs and their functionalities), programming concepts (e.g. storage complexity, taxonomy of functional properties), virtual organisation concepts (e.g. privileges, credentials, ownership), resource negotiation-related concepts (e.g. SLAs), and desired level of QoS-related features (operational costs, elasticity, storage use). In ENTICE, the Knowledge Base will be in charge of storing the taxonomies, thesauri and ontologies used for describing the semantic models of all entities like VMIs and repositories, as well as for linking them with already published ontologies. The ENTICE Knowledge Base as a triple-store should be able to not only retrieve just data, but also to represent meaningful information and knowledge. Key entities and concepts of the developed ontology are shown in Table I.

The knowledge management and information supply play a crucial role in the operation of the overall ENTICE environment. An example where the use of the Knowledge Base is particularly effective is the support for solving some NP-hard optimisation problems, addressed

DOI: 10.1002/cpe

Table I. Important entity concepts modelled in the ENTICE ontology.

ENTITY NAME	DESCRIPTION
DiskImage, VMI, CI	These entities describes representative attributes of VMIs/CIs (e.g. name, version, encryption etc.) and relationship interconnections with entities such as otherDiskImages/Fragments that are predecessors, DiskImageSLA, User, functionality, Quality, Operating system and Pareto. This entity is one of the most relational descriptive entities in ENTICE environment.
Pareto	The entity stores Pareto stage objects (distribution possibilities) for online and offline redistribution that are achieved as a result of MO framework.
Repository	Keeps track of available cloud repositories characteristics.
Fragment	The entity stores preliminary description of fragments.
OperatingSystem	The ENTICE environment should support various types of VMI operating systems.
Data	Some use cases supported in ENTICE environment need the delivery of various content alongside the VMI (e.g. delivery of service calculations in different format types such as images, text files and binaries).
User	Stores information about the users including stakeholder types.
Geolocation	Describes detailed geographical information about the repositories. The detailed geographical information are obtained using the geographical database GeoNames [21].
DiskImageSLA	To support the overall lifecycle of applications deployed as VM images and to facilitate automatic optimization, setup and management in federated Cloud environments, information on the potentially new and resulting Service Level Agreements (SLA) are needed to be stored.
Quality	Stores parameter and result information about VMI synthesis and analysis.
Delivery	This entity specifies the overall VMI deployment event by storing actual QoS, QoE and SLA.

by a combination of MO and Pareto SLA techniques. NP-hard optimisation problem, for example, is to find the optimal distribution of a specific VMI across the distributed repositories, which allows for a specific minimum delivery time at a specific cost.

7.2. Implementation of knowledge management mechanisms

The ENTICE environment can be seen as repository-based system that encapsulates a variety of subsystems. Basically, it is designed to supports Cloud applications with different elasticity requirements, for example, needs to geographically distribute the VM images. The ENTICE technology is decoupled from the Cloud application and its specific details, such as the runtime environment, but it continuously supports the application through optimised VMI/CI creation, assembly, migration and storage. The ENTICE environment inputs are unmodified and functionally complete VMIs or CIs from users. Unlike the other Cloud provider environments in the market, the ENTICE environment transparently tailors and optimises the VMI/CIs for specific Cloud infrastructures with respect to their size, configuration, and geographical distribution, such that they are loaded, delivered (across Cloud boundaries), and executed faster with improved QoS and decreased final cost for the end-users.

DOI: 10.1002/cpe

In relation to the implemented Knowledge Base, the design includes the integration of the subsystems that interconnects with the knowledge base as follows:

- VM Image Synthesis notifies the Knowledge base with all the lifecycle phases of the service including the execution parameters and actual computation times. The knowledge base stores the metadata in the ontology entities in order to assess the approximate time needed to execute operations of the service in the future executions.
- VM Image Analysis uses the knowledge base both, for store all the information about the fragments and also to offer an enriched metadata information through the reasoning mechanisms by identifying redundancies, conflicts and new relationships among the fragments.
- Multi-objective Optimization stores the Pareto metadata in the knowledge base and exploits the reasoning mechanisms of the knowledge base to optimize the MO process. To be more concrete, data set needed for the MO is filtered in the knowledge base through reasoning mechanisms (e.g. fragment characteristics, geolocation information etc.) and SPARQL purposive queries. By doing that the filtered results reflect a subset of best possible solutions that significantly reduces the MO execution time.
- VMI Assembly mainly retrieves all needed information of an VMI needed to be constructed from the fragments. A crucial research aspect of the knowledge base is how to use the available ontology information (e.g. SLA, bandwidth, cloud characteristics, fragment positioning etc.) to propose an optimal assembly plan for the VMI assembly service.

In general the knowledge base as a component is presented as a REST service based on Jersey framework [20]. The main knowledge base technology is Jena Fuseki, a RDF database described with Web Ontology Language (OWL). Among Fuseki available framework functionalities additional reasoning tools needed to be integrated (e.g. HermiT, Pellet etc.) in order to fully exploit data manipulation, data validation, introduction of new individual relational characteristics and other mechanisms.

8. EXPERIMENTAL EVALUATION

In this section a broad experimental evaluation of the researched methods and techniques for efficient VMI management in federated Cloud environment has been presented. The behavior of the developed modules has been studied though a set of evaluation indicators, specifically selected for assessment of each individual VMI operation. To this end, the evaluation activities have been conducted from two distinctive aspects, covering the assessment of the: (i) VMI synthesis and analysis module, (ii) multi-objective VMI redistribution and replication module. All evaluation processes were performed with tight interaction with the knowledge base and reasoning mechanism.

The evaluation experiments were conducted on a functional test-bed environment, distributed across various geographical locations. The VMI synthesis and analysis module has been deployed at the premises of the Hungarian Academy of Science, which provided 4 physical nodes, totaling 176 physical cores and 128 GB of RAM memory per node. Furthermore, the Multi-objective framework was located at premises of the University of Innsbruck, which includes 7 physical nodes, totaling 168 physical cores and 32 GB RAM per node. Lastly, the Knowledge base has been located at the University of Ljubljana. This location includes, 4 physical nodes, each one offering 8 physical cores and 8 GB of RAM per node. The physical interconnection between the modules has been established over dedicated optical links, while the logical communication between the modules was performed by utilizing RESTful and SOAP protocols.

8.1. VMI synthesis and analysis module evaluation

As introduced in Section 5, VMI synthesis offers image maintenance operations. In order to evaluate this feature of the ENTICE environment, we used two simpler scenarios provided by the industrial

DOI: 10.1002/cpe

partners of the project. The first one is performed on the image of WordPress 4.3.1 on CentOS 7.2, and the second one is performed on the image of Redmine on CentOS 6.8, provided by the ENTICE industrial partners Wellness Telecom and FlexiOPS respectively. Previously, our partners built their VMIs manually in an error prone and time consuming way.

	WordPress 4.3.1 (CentOS 7.2)	**Redmine (CentOS 6.8)**
Original image contents (Bytes)	1 548 718 080	2 273 243 136
Recipe-based image contents (Bytes)	1 995 644 433	1 550 196 435
Optimized image contents (Bytes, 5 iterations)	860 170 897	684 330 624
Optimized image size (Bytes, QCOW2 format)	1 269 628 928	1 146 617 856
Original build time (minutes, estimated)	60	60
Recipe-based build time (minutes)	17	17

Table II. Details of images used for evaluation: before and after optimization

Detailed properties of these images can be seen in Table II. It states the sizes of the contents of the considered images in terms of: the original image, recipe-based created image and the optimized image. According to these results we can see that the contents of the WordPress image could be reduced to 55%, while the contents of the Redmine image to 30% after five iterations of optimization steps. The build time of the original and recipe-based images are also shown here.

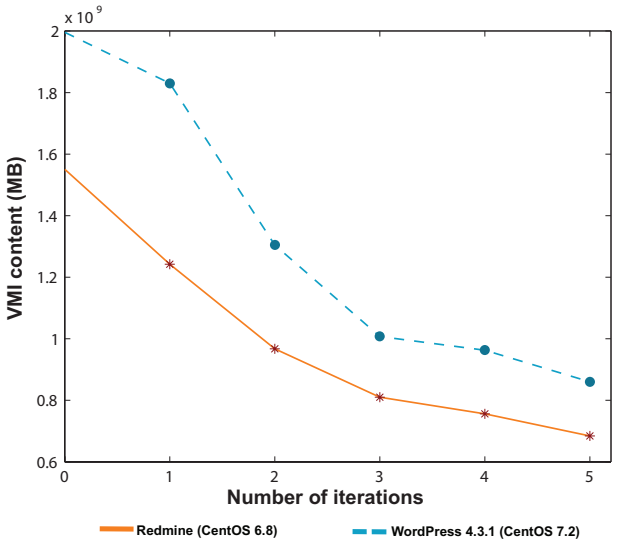


Figure 6. Wordpress and Redmine optimization progress: image contents in million Bytes (MB) after the different optimization iterations (0-5), where iteration 0 represents the recipe-based built image

Figure 6 presents the results of the iterative target size-optimization. Here we can see how the content sizes of the images were iteratively reduced during the optimization steps for the two cases. The results show that the first three iterations performed higher amount of content reduction, while the last two could only improve with a smaller amount of additional bytes.

8.2. Multi-objective VMI repository optimization framework evaluation

To begin with, the behaviour analysis and performance evaluation of the VMI optimization framework require various different aspects to be taken into account. The essential feature that this framework offers beyond the current state-of-the-art is the possibility for replication of the

DOI: 10.1002/cpe

most frequently used VMIs during the process of re-distribution. To this end, we have initially focused the evaluation activities on identifying the optimal value of the replication coefficient k in regards with the computational performance. As described in Section 6, the replication coefficient k controls the number of VMIs that have to be replicated, thus in case of $k = 1$, or full replication of the 50% most frequently used VMIs, the complexity of the multi-objective optimization algorithm can be increased multi-fold. Figure 7 provides an insides on the execution time of the optimization algorithm in regards to multiple different values of k. For the current evaluation scenario a synthetic benchmark data was created and stored in the knowledge base, including 100 repository sites and computational Clouds, and identical number of VMIs. The optimization algorithm, was configured with an initial population of 100 individuals and 10000 separate evaluations/generations.

The evaluation results, clearly show that in the cases when the replication coefficient is below 0.5, the execution time is increased by only 10%. This implies, that if it is required for the 25% of the most frequently used images to be replicated, the effect on the optimization execution time will be minimal.

As we are dealing with an evolutionary multi-objective algorithm, it is crucial to asses the algorithm's scalability in regards with the number of separate evaluations of the individuals within the population size. Figure 8 provides a comparison between the execution times of the optimization framework without support for replication [2] and with replication support. The benchmark data and population size used in this test were identical to the previous one. Both, optimization algorithms scale linearly, with a relatively constant performance difference of around 40% in the cases when no replication is performed.

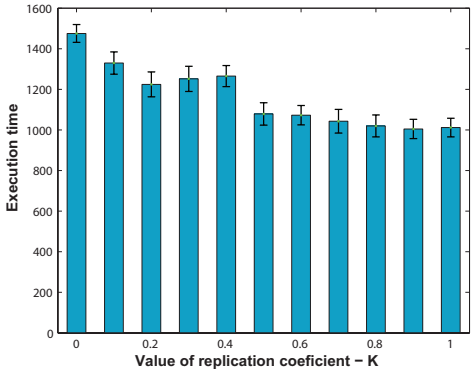


Figure 7. Optimization algorithm execution time for different values of k

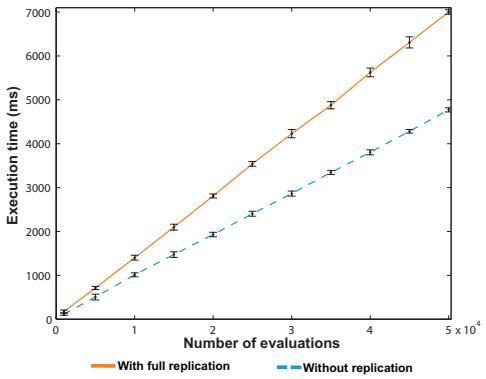


Figure 8. Optimization algorithm execution time for different number of individual evaluations

Furthermore, the optimization framework with replication support has been evaluated on the bases of the quality of the Pareto optimal solutions for different values of the genetic parameters, such as number of individuals and number of evaluations/generations. The assessment was conducted for a specific scenario enclosing the re-distribution of 100 VMIs with replication coefficient $k = 1$. The hypervolume indicator [13] has been used to compare the quality of the separate solutions provided during the experiments. Figure 9 presents a comparison of the mean value and standard deviation of the quality of the solutions in contrast to the number of utilized individuals and performed evaluations per experiment. By analysing the results, it can be concluded that in the cases when low computational time is required, it is essential to reduce both the number of individuals and performed evolutions, thus guaranteeing best computational performance to quality ratio. Contrary, when the computational time is not and issue, like in the case of off-line VMI redistribution, higher populations sizes with increased number of evolutions should be used. For example, in the case when 100 individuals are being utilized, we can expect best quality of the solutions when less then 5000 separate individual evaluations are performed. Above this threshold, the low number of individuals can easily lead to low diversity and reduced search capabilities, thus

DOI: 10.1002/cpe

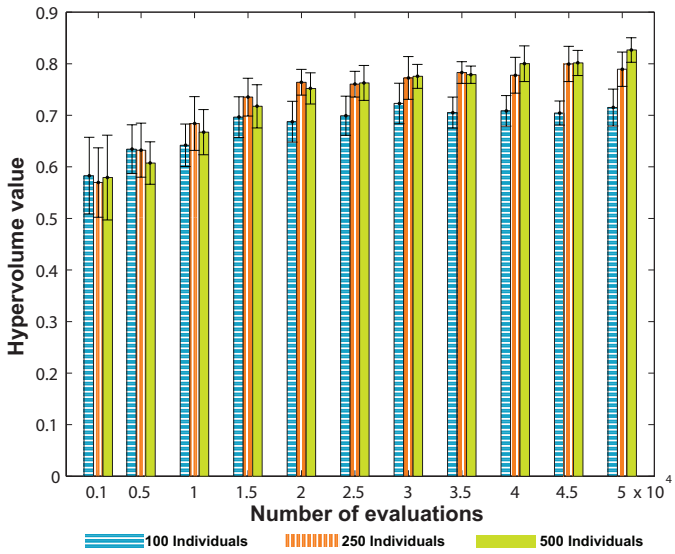


Figure 9. Quality of the optimal solutions in contrast with the number of individuals

producing solutions with limited hypervolume values. In the case of ENTICE, the re-distribution and replication are performed off-line, thus requiring high population size and increased number of evolutions. To better illustrate the difference in quality among the solutions, on Figure 10 we present a comparison between the best solutions gathered from the experiments with 100 and 500 individuals.

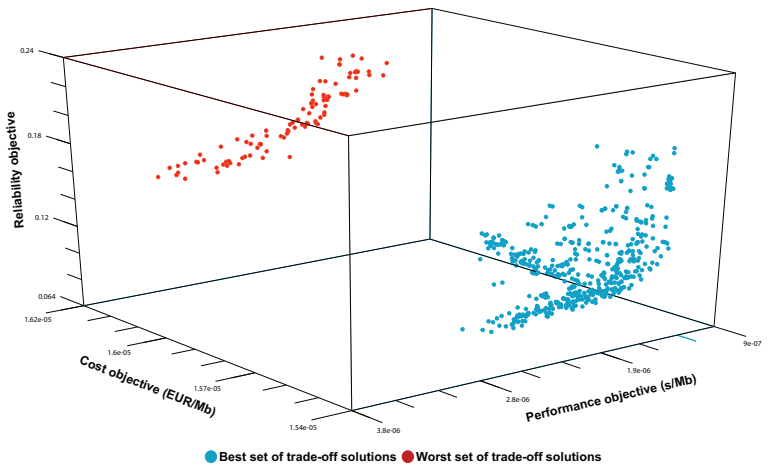


Figure 10. Visualized comparison of the best and worst solutions provided by the optimization framework

Lastly, in Table III we provide a comprehensive review of the exact objective values for the optimal trade-off distribution solutions calculated by the optimization framework. Moreover, a comparison has been presented with a set of mapping solutions determined by using non-optimized ”round robin” mapping model for storing VMIs, and a multi-objective re-distribution algorithm without replications support [8]. The cost objectives have been calculated based on the publicly provided price list for storing data in the Cloud by public companies, such as Amazon S3 and Microsoft Azure. The performance objective has been modelled based on the reported communication performance measures for 10Gbit and 1Gbit Ethernet [16]. For optimization reasons, the communication bandwidth values, were represented as delivery time needed for 1Mbit

DOI: 10.1002/cpe

of data to be transferred from the source repository to the computational Cloud within the ENTICE environment. The system reliability values, were synthetically generated based on real values provided by multiple public Cloud providers, and do not include any other replication strategies.

The results indicate very high efficiency of the VMI re-distribution framework with replication support, as it can provide higher quality mapping solutions, especially in regards with the performance and reliability objective, yielding improvements of 51% and 74% respectively. On the other hand, the cost requirements can be increased by up to 46% if full replication is required. Using lower replication coefficient, such as $k = 0.5$, can diminish this issue and provide almost 50% performance increase, while incurring only 11% increase in financial costs.

	Round-robin	**No-replication**	**Replication** $k = 0.5$	**Replication** $k = 1$
Cost objective (EUR/MB)	0.00001612	0.00001569	0.00001805	0.00002360
Difference-Cost (%)	/	-2.66638854	11.96255695	46.36499402
Performance objective (s/Mbit)	0.00000285	0.00000162	0.00000142	0.00000137
Difference-Performance (%)	/	43.13983662	50.00573665	51.94813878
Reliability objective (1-System R.)	0.00000285	0.00000162	0.00000159	0.00000137
Difference-Reliability (%)	/	28.61160910	42.08287635	74.05835564

Table III. Comparison between different VMI re-distribution strategies

9. CONCLUSIONS

In order to address the challenges that arise with the adoption of federated Cloud environments, in this paper a set of novel technologies and design approaches for efficient operation of distributed VMI repositories has been present. To accomplish this rather ambitious goal, extensive research was conducted on the current state-of-the-art and distinctive software modules have been developed to support the following operations: (i) simplified creation of light weight and highly optimised VMIs tuned for specific application requirements; (ii) VMI repository optimization based on multi-objective approach; and (iii) efficient reasoning mechanism for streamline support of complex VMI operations.

The introduced VMI management techniques have been implemented, integrated and evaluated as essentials elements of the ENTICE environment. Based on the evaluation results, it can be concluded that the proposed VMI synthesis and analysis technique can reduce the size of the VMIs by up to 55%, while trimming the image creation time by 66%. Furthermore, the repository optimization framework, can reduce the VMI delivery time by up to 51% and cut down the storage expenses by 3%. Moreover, by implementing optimized replication strategies, the framework can increase the system reliability by 74%, even in the cases when no other reliability strategies are implemented. All these processes, have been supported by efficient knowledge base and reasoning mechanism, which can reduce the communication time between the separate modules within the ENTICE environment to bare minimum.

Regarding the future work, we plan to extended the current environment to support fragmentation and re-assembly of VMIs based on similar functional blocks, thus allowing more optimal VMI storage and distribution. 01

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 644179 (ENTICE project: dEcentralised repositories for traNsparent and efficienT vIrtual maChine opErations).

REFERENCES

1. Iosup Alexandru, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema, *"Performance analysis of cloud computing services for many-tasks scientific computing"*, IEEE Transactions on Parallel and Distributed systems 22, no. 6 (2011): 931-945.
2. Dragi Kimovski, Nishant Saurabh, Sandi Gec, Polona Štefanič, Gabor Kecskemeti, Vlado Stankovski, Radu Prodan and Thomas Fahringer, *"Towards an Environment for Efficient and Transparent Virtual Machine Operations:The ENTICE Approach"*, IEEE 4th International Conference on Cloud Networking (CloudNet), Pisa, Italy, 2015.
3. Goiri, I., Guitart, J., Torres, J., *"Characterizing cloud federation for enhancing providers' profit"*, 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 123-130.
4. A. Celesti, F. Tusa, M. Villari, and A. Puliafito, *"How to enhance cloud architectures to enable cross-federation"*, IEEE CLOUD 2010, pp. 337–345.
5. P. C. Brebner, *"Is your cloud elastic enough?: Performance modelling the elasticity of infrastructure as a service (iaas) cloud applications"*, Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ICPE 2012, ACM, pp. 263–266.
6. Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio, *"An Updated Performance Comparison of Virtual Machines and Linux Containers"*, RC25482 (AUS1407-001) July 21, 2014, Computer Science
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T, *"A fast and elitist multiobjective genetic algorithm: NSGA-II" Evolutionary Computation"*, IEEE Transactions on, 6(2), 182-197.
8. Dragi Kimovski, Nishant Saurabh, Sandi Gec, Vlado Stankovski and Radu Prodan, *"Multi-objective Optimization Framework for VMI Distribution in Federated Cloud Repositories"*, Large Scale Distributed Virtual Environments, LSDVE 2016 in conjunction of Euro-Par 2016, Grenoble, France.
9. Diaz, G. von Laszewski, F. Wang, A. Younge, and G. Fox, *"Futuregrid Image repository: A Generic catalog and Storage System for Heterogenous Virtual Machine Images"* Third IEEE International Conference on Coud Computing Technology and Science (CloudCom2011), 2011.
10. J.V. Carrión, G. Moltó, C. De Alfonso, M. Caballer, and V. Hernández, *"A Generic Catalog and Repository Service for Virtual Machine Images"*, 2nd International ICST Conference on Cloud Computing (CloudComp 2010).
11. Kaveh Razavi, Liviu Mihai Razorea, and Thilo Kielmann, *"Reducing VM Startup Time and Storage Costs by VM Image Content Consolidation"*, 1st Workshop on Dependability and Interoperability In Heterogeneous Clouds, Euro-Par 2013: Parallel Processing Workshops,2013.
12. Di Wu, Yupeng Zeng, Jian He, Yi Liang, and Yonggang Wen, *"On p2p mechanisms for vm image distribution in cloud data centers: Modeling, analysis and improvement"*,CloudCom 2012, pp. 50–57. IEEE Computer Society, 2012.
13. Zitzler, Eckart, Dimo Brockhoff, and Lothar Thiele, *"The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration"*, international Conference on Evolutionary Multi-Criterion Optimization. Springer Berlin Heidelberg, 2007.
14. Sandi Gec, Dragi Kimovski, Radu Prodan, and Vlado Stankovski, *"Using constraint-based reasoning for Multi-Objective Optimisation of the ENTICE environment"*, IEEE The 12th Semantics, Knowledge and Grids on Big Data (SKG2016). Beijing, China, 2016.
15. Matthias Schmidt, Niels Fallenbeck, Matthew Smith, and Bernd Freisleben, *"Efficient distribution of virtual machines for cloud computing"*, in Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP '10, pages 567–574, Washington, DC, USA, 2010. IEEE Computer Society.
16. Feng W. C., Balaji P., Baron C., Bhuyan L. N., Panda D. K., *"Performance characterization of a 10-Gigabit Ethernet TOE"*, High Performance Interconnects, 2005. Proceedings. 13th Symposium on (pp. 58-63). IEEE.
17. Kecskemeti, G., Marosi, A. Cs., Kertesz, A. *"The ENTICE approach to decompose monolithic services into microservices"*. In High Performance Computing & Simulation (HPCS), 2016 International Conference on (pp. 591-596). IEEE.
18. Image Factory - http://imgfac.org/
19. Packer - https://www.packer.io/
20. Jersey - RESTful Web Services in Java: https://jersey.java.net/
21. GeoNames - http://www.geonames.org/
22. VMWare - http://www.vmware.com/appliances
23. Science Cloud - http://scienceclouds.org/marketplace
24. Amazon Image Service - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

DOI: 10.1002/cpe