ICT 269978

Integrated Project of the 7th Framework Programme

COOPERATION, THEME 3
Information & Communication Technologies
ICT-2009.5.3, Virtual Physiological Human

Work Package: WP2

Data and Compute Cloud Platform

Deliverable: D2.1

Analysis of the State of the Art; Work Package Definition

Version: 1.3

Date: 27/05/2011

Document Information

IST Project Num	FP7 – ICT - 269978	**Acronym**	VPH-Share
Full title	Virtual Physiological Human: Sharing for Healthcare – A Research Environment		
Project URL	http://www.vphshare.org		
EU Project officer	Joël Bacquet		

Work package	**Number**	2	**Title**	Data and Compute Cloud Platform
Deliverable	**Number**	2.1	**Title**	Analysis of the State of the Art; Work Package Definition

Date of delivery	**Contractual**	2011-05-31	**Actual**	2011-05-31
Status	Version 1.3		Final ☒	
Nature	Prototype ☐ Report ☒ Dissemination ☐ Other ☐			
Dissemination Level	Public (PU) ☒ Restricted to other Programme Participants (PP) ☐ Consortium (CO) ☐ Restricted to specified group (RE) ☐			

Authors (Partner)	Marian Bubak, Tomasz Bartyński, Marek Kasztelnik, Maciej Malawski, Jan Meizner, Piotr Nowakowski (CYFRONET) Spiros Koulouzis (UvA) David Chang, Stefan Zasada (UCL) Enric Sarries (AOSAE)			
Responsible Author	Piotr Nowakowski		**Email**	p.nowakowski@cyfronet.pl
	Partner	CYFRONET	**Phone**	n/a

Abstract (for dissemination)	This deliverable details the state of the art in research area relevant to the development and integration work which is to occur in Work Package 2 throughout the lifecycle of the VPH-Share project. It also presents a general overview of the key challenges involved in developing a cloud platform for VPH-Share and a conceptual overview of how WP2 aims to overcome these challenges.
Keywords	cloud platforms, PaaS, IaaS, distributed systems, cloud resource management, distributed data access

Version Log			
Issue Date	**Version**	**Author**	**Change**
2011-05-15	0.1	Piotr Nowakowski (ed.); contributing authors from CYFRONET, UvA, UCL, AOSAE	First version for internal review
2011-05-25	1.0	Norman Powell, Steven Wood, Piotr Nowakowski	Updates and changes following internal review
2011-05-27	1.3	Jan Meizner, Enric Sarries, Piotr Nowakowski	Further updates; revised section on cloud security

Contents

List of Figures

EXECUTIVE SUMMARY

This document fulfils a dual purpose: it defines the work to be done in Work Package 2 of the VPH-Share project by providing a general overview of the aims of this WP and the means by which WP2 plans to meet these goals, and it also contains an in-depth description of the state of the art relevant to the research and development tasks in WP2. More specifically, the deliverable can be divided into an introductory section, which establishes some basic concepts and presents the basic use cases of interaction with WP2 mechanisms and tools, and a state of the art analysis section, which lists the existing solutions and ongoing work in the following areas:

- deployment and execution of applications in cloud infrastructures;
- access to high performance computing (non-cloud) infrastructures;
- access to large binary data in the cloud;
- data integrity, availability and retrievability;
- security aspects related to cloud computations.

The analysis presented in this document is meant as a starting point for the development of a detailed WP2 architecture, which is expected to coalesce by Project month 6 and will be described in a separate WP2 deliverable (D2.2). Thus, each section comes with specific recommendations for developers of VPH-Share tools and suggests specific technologies and components to be adopted by WP2. In general, the following conclusions can be drawn from to-date analysis:

- Regarding access to computational resources, we need to be able to deploy applications in public clouds as well as on private cloud infrastructures. For reasons detailed in Section 4.1 It seems advisable to retain compatibility with AWS (Amazon Web Services) solutions while using OpenStack as a management platform for private (project-oriented) cloud installations;
- Access to legacy HPC infrastructures can be ensured by integrating an Application Hosting Environment (AHE) client in the virtual machines on which VPH-Share applications are to be hosted;
- Data access should be provided by way of an OCCI (Open Cloud Computing Interface) client facilitating storage and retrieval of binary data objects in various underlying (cloud-specific) repositories;
- A project-wide registry of managed binary data objects has to be maintained and an automated Atomic Service should exist to periodically check the integrity of such objects, migrate them between storage resources (upon request) and provide direct access with the use of the VPH-Share Master Interface (developed by WP6);
- All of the above components must implement a common security policy, enforced with the use of project-wide security tokens, issued by a trusted authority and accepted by all VPH-Share modules which expose an external (SOAP or REST) interface.

This document is meant as a live deliverable – should additional technologies become relevant to WP2 development at the design/implementation stage, we intend to issue extensions to the topics discussed here, taking into account ongoing developments.

1 INTRODUCTION

The goal of Work Package 2 (Data and Compute Cloud Platform) is to develop, integrate and maintain an environment which will enable the VPH-Share workflows, as well as any application making use of VPH-Share resources, to operate on top of the cloud and high-performance computing infrastructure provided by the project.

In order to fulfill this task, Work Package 2 needs to deliver a consistent service-based system which will enable end-users to deploy the basic components of VPH-Share application workflows (known as Atomic Services) on the available computing resources, and then enact workflows using these services. Both types of activity need to be supported concurrently – note that the developer of an application workflow is not necessarily the person interacting with the workflow once it has been prepared. Thus, the end-user interfaces (and – by extension – the Work Package 2 services which support them) must cater to both groups of users. This division of responsibilities will be further elaborated upon in Section 2.

Given the above requirements, the primary aim of this document is to provide a basis for the development of a detailed architecture of WP2 services. We will investigate the current state of the art in distributed computing and binary data access solutions, and formulate recommendations for VPH-Share. The document is structured as follows:

- Section 2 outlines the purpose of Work Package 2 and shows how WP2 fits into the overall vision of VPH-Share. It also discusses generic use cases, explaining how the services provided by WP2 will be used in support of the actions performed by end users of Project software. Finally, it establishes some basic concepts and terms which will subsequently be used to describe the features of WP2 software.
- Section 3 lists the key issues which arise from the use cases discussed in Section 2, setting the stage for a detailed description of the state of the art with regard to specific areas of interest.
- Section 4 presents the current state of the art in each of the fields identified in Section 3, listing ongoing scientific and commercial initiatives and providing recommendations for the VPH-Share infrastructure.
- Section 5 summarizes the presented descriptions and contains general conclusions.

2 WORK PACKAGE DEFINITION AND HIGH-LEVEL OVERVIEW

Work Package 2 is responsible for maintaining the interface between the end-user tools, developed (among others) in WP6 and the actual computing and data storage resources, as depicted in Figure 1. According to the Project’s Description of Work [1], the WP2 approach is not to develop low-level cloud middleware or services, but rather to build on top of existing solutions. Thus, we intend to use available IaaS providers (e.g. Amazon EC2) together with privately-deployed open source cloud platforms (e.g. Eucalyptus, Nimbus). The proposed solutions will be positioned above such IaaS layer, tailoring for collaborations which either have access to local resources or intend to outsource the infrastructure to commercial providers.

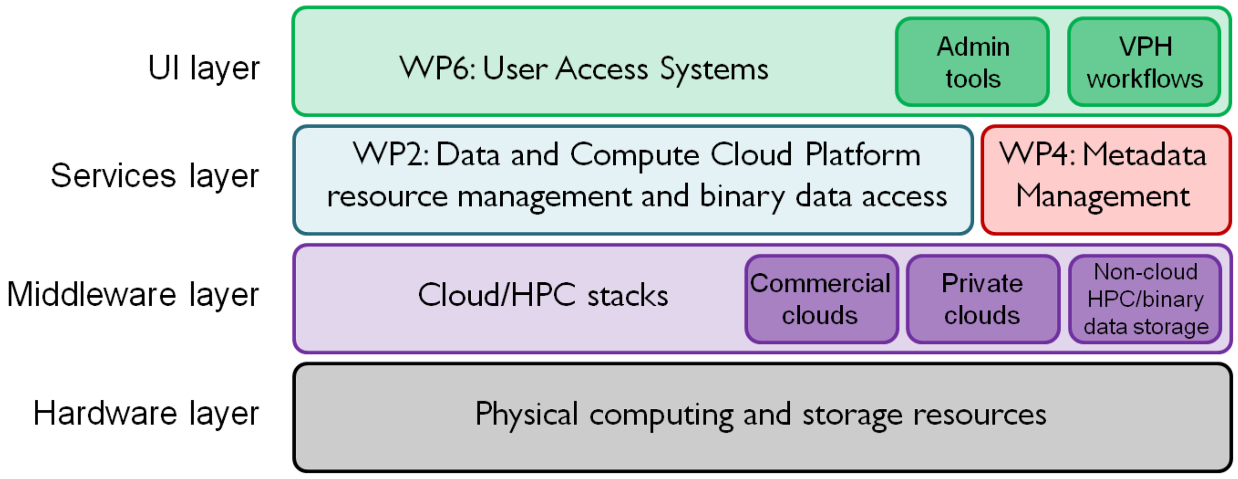

Figure 1: WP2 in the VPH-Share architecture

The services developed in Work Package 2 will provide a basis for higher-level services and tools developed in other technical Work Packages of the Project. The computational workflows from WP5 will be integrated with the platform and the user tools (WP6) will provide access to the services. Hence, in order to provide a consistent basis for the development and integration of Project middleware, our work needs to be based upon in-depth analysis of the state of the art in distributed infrastructures for high-performance computing and large-scale data management. Moreover, due to the sensitivity of data which will be managed with the use of VPH-Share tools, we need to devote special attention to mechanisms ensuring security and reliability of any data stored in the resulting infrastructure. In addition to purely technical considerations, we also have to take into account the requirements for the VPH-Cloud cloud platform resulting from Task 2.7, where all VPH flagship workflow leaders will contribute the descriptions of their applications and services. Such combined approach of "application pull" and "technology push" will result in the design and the first prototype which will be subject to iterative evaluation and refinement throughout the remainder of the Project's life cycle.

In line with the considerations presented above, we will now outline the process by which Work Package 2 services will be exploited in support of the basic use cases of the VPH-Share project. While a specific, in-depth description of each WP2 module will be included in the second WP2 deliverable (due by Month 6 of the Project), the aim of this introductory session is to give the reader an overall impression of how Work Package 2 fits into the general VPH-Share vision and which areas need to be covered by the state of the art analysis in order to ensure that the WP2 objectives are met.

2.1 Deployment of a VPH-Share application workflow

As mandated by the Description of Work [1], the VPH-Share workflows consist of Atomic Services, encapsulating domain-specific functionality of the VPH-Share applications. In order to ensure smooth deployment of the pilot applications in the VPH-Share infrastructure, WP2 must deliver tools which will expose the features of atomic services (as defined in DoW) while ensuring that application developers remain unconstrained with respect to the technologies in which their applications are developed. In short, WP2 must enable deployment of arbitrary applications on the available cloud and HPC resources. The IaaS [2] model, also called a *Resource Cloud*, is naturally suited to this type of integration as instead of predefined "building blocks" from which applications can be constructed (such as services or components) it instead provides users with a platform on which to deploy their own software. The purpose of this section is to outline how VPH-Share applications will be constructed and deployed, as well as to firmly establish certain concepts which will be consistently applied throughout the remainder of the Project.

The deployment use case (i.e. preparation of a VPH-Share application for use) is briefly illustrated in Figure 2. It should be noted that this task falls to the application developer, i.e. a person intimately familiar with the implementation specifics of a given application. This action needs to occur once per lifetime of the VPH-Share application. Once completed, the application can be accessed by end users numerous times with no further developer involvement.

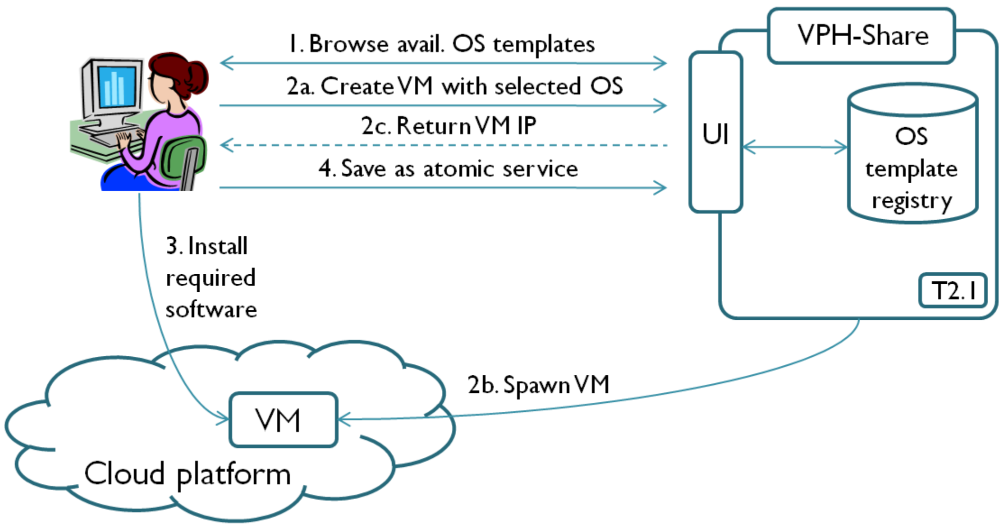
1. Browse avail. OS templates
2a. Create VM with selected OS
2c. Return VM IP
4. Save as atomic service
VPH-Share
UI
OS template registry
T2.1
3. Install required software
2b. Spawn VM
VM
Cloud platform

Figure 2: Creation of an Atomic Service on the basis of an existing application, component or standalone tool

As can be seen, the application developer's interaction with the VPH-Share environment begins with the Master UI delivered by WP6 (step **1**), where the developer requests the creation of a new atomic service (step **2a**). In response, the WP2 component responsible for AS management spawns (step **2b**) and exposes (step **2c**) a fresh Virtual Machine, residing in the cloud. Upon receiving the IP (and administrative login information) for this VM, the developer may proceed to install the software belonging to his/her application (step **3**). Once complete, this machine can be registered with VPH-Share mechanisms whereupon it becomes an atomic service (step **4**). The project infrastructure stores information on atomic services in its registry (maintained by Work Package **4**) and can provide it to end users as well as automated tools such as the Workflow Composer. In this way, atomic services are managed by the VPH-Share infrastructure. It should be noted that, at least as far as cloud computing resources are concerned, operating an instance of an atomic service, deployed and running on leased computing resources, can be expensive as it would require the Project to finance such resources whether they are exploited or not. Thus, we envision a situation where atomic services are stored in the form of virtual system images and instantiated on demand, as requested by end users or workflow enactment middleware. We therefore define the following concepts, which will be applied throughout this deliverable and over the course of WP2 development:

- **Virtual Machine**: A virtual host spawned from a preconfigured operating system template and exposed to application developers so that they can install and deploy their applications (or components thereof). Virtual machines make no assumptions regarding the technologies with which applications are developed – the developer obtains administrative login information for his/her virtual machine and can deploy arbitrary software, libraries, modules etc. It is also possible to preinstall certain software on virtual machines, if required by the VPH-Share infrastructure (for instance, security modules, client service wrappers etc.)
- **Atomic Service:** A Virtual Machine on which components of VPH-Share-specific application software have been installed, wrapped as a virtual system image and registered with VPH-Share management tools provided by Work Package 2. Note that atomic services are not necessarily instantiated, i.e. they should be treated as *images* rather than actual network hosts. WP2 takes care of instantiating such services as required by the end users, leading to the concept of an Atomic Service Instance, presented below. An important assumption is that the atomic service exposes its functionality through a web service interface, mediated by the Virtual Machine's web server software (such as Apache [3]).
- **Atomic Service Instance**: A specific atomic service deployed on computing resources (cloud-based or otherwise) and providing VPH-Share application functionality through a web service (SOAP or REST) interface, typically in the form of a publicly-accessible host exposing a WS API (which is also registered with VPH-Share mechanisms and can be queried by developers). Atomic Service Instances need to be secured against unauthorized access through a token-based security mechanism, developed in Task 2.6.

While a single Virtual Machine is always used to create and store exactly one atomic service, a number of separate instances can be launched for a single atomic service. The decision when to deploy instances (and how many of them should be launched) rests with the WP2 deployment planning and execution environment. This concept is briefly outlined in Figure 3.

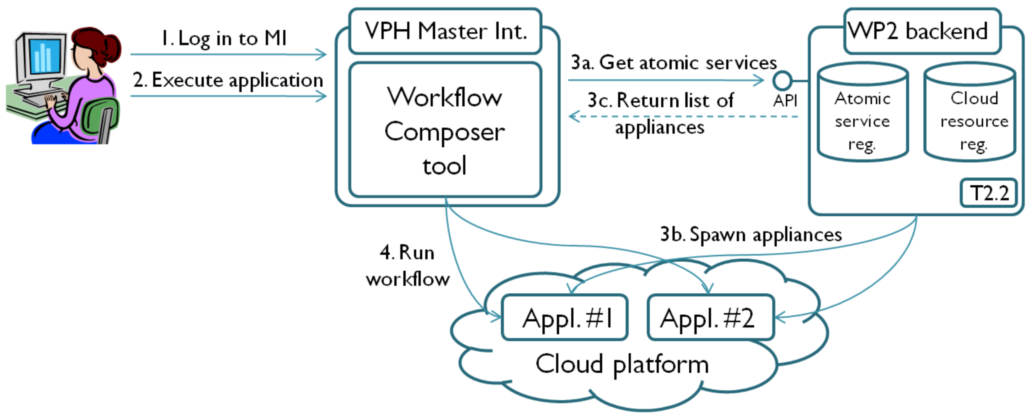

Figure 3: Deployment of VPH-Share Atomic Service Instances

As can be seen, the Workflow Composer tool, accessible through the VPH-Share Master Interface, is responsible for informing the WP2 backend which atomic services are required to execute a given workflow. The backend responds by launching the required atomic service instances and passes a list of instances back to the WFC, which can then commence executing the workflow.

2.2 Classes of Atomic Services

It is worth noting that several different classes of atomic services may coexist in the VPH-Share infrastructure. More specifically, three distinct classifications can be introduced, as outlined in the following subsections.

2.2.1 Resource requirements

In terms of the underlying resources needed to provide the functionality of atomic services (and their instances), the following distinction can be made:

- **Cloud-based Atomic Services**: these atomic services are based on cloud resources and are deployed either in private clouds or in public cloud infrastructures made available to the Project. Specifically, in order to create instances based on these atomic services, the WP2 tools need to interact with the underlying resources through one or more of the available cloud computing stacks, as discussed in section 4.2.3).
- **Non-cloud Atomic Services**: a separate category of atomic services which do not operate on cloud resources but nevertheless expose the same API to the WP2 management tools. This category specifically includes HPC clients developed within Task 2.3 and may also include plugins for other (legacy) non-cloud computing platforms. Section 4.3 contains more information regarding such services.

2.2.2 Genericity

In terms of the actual *purpose* of each atomic service, the following classes can be distinguished:

- **Application-Specific Atomic Services**: These atomic services will be specifically constructed as parts of VPH-Share application workflows, as reported in [1]. An atomic service may encapsulate an entire VPH-Share application or parts thereof – the decision which approach to pursue rests with workflow developers. The purpose of each of these services is to provide application-specific functionality for the workflows which are prepared and enacted using WP6 tools on behalf of the end users of the VPH-Share platform.
- **Generic Atomic Services**: These atomic services are associated with the platform itself rather than with any particular application. They enable end-users as well as automated VPH-Share tools to perform administrative tasks, ensure security, support post-processing of application results (to enable their visualization in the Master Interface) and facilitate automatic maintenance of VPH-Share datasets (including direct access to data mediated by Task 2.4, and data availability/provenance tracking within Task 2.5). Generic atomic

services can be exploited directly by the WP6 user interfaces or indirectly, by VPH-Share application developers who can embed them in their application workflows.

2.2.3 Persistence

From the point of view of their persistence, atomic service instances can be divided into the following categories:

- **Instances deployed on demand**: as mentioned above, most atomic service instances will need to be instantiated on demand so as to conserve cloud resources. In such cases, WP2 management tools will be able to spawn an instance for a given atomic service when requested by WP6 workflow management, and then despawn it once a given workflow has concluded. We foresee this as the primary mode of operation for application-specific atomic services.
- **Persistent instances**: in certain cases – such as for generic atomic services which perform regularly-scheduled maintenance tasks or are otherwise time-critical (and cannot tolerate delays associated with instantiating a fresh atomic service) – WP2 will be able to maintain a given atomic service in an “always ready” state by preinstantiating an instance which is persistently deployed on computing resources and can accept incoming requests. Clearly, the need for responsiveness will have to be balanced against the additional cost incurred by maintaining such an instance– the decision on whether to spawn static instances will have to be made by the WP2 tools on the basis of load-balancing algorithms, or delegated to the system administrator with the use of WP6 interfaces.

Having briefly outlined how WP2 will support the requirements imposed upon it by the Description of Work we can proceed with a more detailed explanation of the specific areas of responsibility for WP2 technical tasks and the technologies which fall within the scope of each of these tasks.

3 Key Challenges Involved in Developing a Cloud Platform for VPH-Share

In order to provide a platform which supports management and provisioning of cloud and HPC resources consistent with the above description, Work Package 2 needs to build upon existing resource management technologies. We intend to apply industry standards whenever possible and extend them where required by the specifics of VPH-Share. Thus, we intend to focus our analysis of the state of the art on the following fields.

3.1 Management of cloud resources

Cloud computing is a relatively fresh concept in information technology and a number of approaches and competing platforms have recently appeared on the market (see Section 4.1.3 for an in-depth discussion of these platforms). Since no global standards regarding the management and integration of cloud resources have been codified thus far, we need to evaluate the *de facto* industry standards and choose the most promising solutions while making sure that we do not subject ourselves to “vendor lock-in” by choosing a single platform to the exclusion of all others. This leads to the following questions, which will need to be addressed further on in this deliverable:

- What technologies are currently being used to expose and access cloud resources in IaaS solutions available on the open market? What are their advantages and constraints? To what extent do they permit fine-grained management of cloud resources? What are their associated security and billing models?
- What middleware platforms exist for management of private cloud installations? Do they ensure cross-compatibility with public computing clouds?
- What is the relative maturity of the technologies discussed above? Are there any legal issues which would preclude VPH-Share from making use of these technologies?

3.2 Management of HPC resources

In addition to managing cloud resources and deploying workflows on such resources, VPH-Share will also need to ensure compatibility with existing HPC infrastructures. Thus, a separate section of the document needs to be dedicated to reviewing the state of the art in non-cloud HPC platforms, which involves answering the following questions:

- What other (non-cloud) distributed computing infrastructures are currently being exploited within the scientific community? What are the requirements (technical and otherwise) for accessing such infrastructures?
- How can arbitrary code (i.e. atomic services) be deployed on non-cloud HPC resources? How to ensure consistent management of such code along with cloud-based resources?

3.3 Access to Binary Data

Work Package 2 is tasked with providing access to binary data in parallel with atomic services and their associated instances. This task calls for a separate study aimed at describing and selecting optimal data management solutions from the point of view of VPH-Share requirements. Important issues include:

- What tools and technologies are provided by existing cloud computing stacks (both public and private) for the purposes of storing and manipulating binary data? To what extent does the owner of binary data have control over where such data is physically stored and who is permitted to access it?
- What access protocols are in use to store and access binary data? Are these protocols consistent with the access restrictions foreseen in the scope of medical data management in the VPH-Share project?
- What security mechanisms are used to protect binary data? How can the Project ensure that such data is not leaked, either by way of unauthorized use of VPH-Share tools or through malicious access to physical resources by the infrastructure provider?
- What guarantees can be provided to end users of the VPH-Share platform regarding consistency and availability of binary data in the cloud? How to ensure that these guarantees are met?

3.4 Security

Security is an issue which should be addressed in parallel to the functional characteristics of the WP2 infrastructure. While each of the topics outlined above necessarily touches upon security issues, ensuring consistent security enforcement throughout the Project requires us to devote a separate section to the concept of securing distributed computing systems (of which VPH-Share is a prime example). Thus, the following questions arise:

- What security mechanisms are associated with *de facto* cloud computing and HPC standards and how can these be integrated with one another to provide a uniform security layer for VPH-Share?
- How to ensure secure enactment of the use cases outlined in Section 2.1 and how to make sure that each component of the distributed computing platform (including VPH-Share tools, atomic services and their instances) is appropriately secured against unauthorized access?
- How to integrate security in a way which does not place undue burden on the end users of VPH-Share application workflows? Is a Project-wide Single Sign-On mechanism feasible given the security requirements imposed by individual Tasks?

The remainder of the deliverable is intended as an in-depth analysis of each of the aspects presented in this section. In addition to discussing the current state of the art related to cloud, HPC and binary data management, we will also discuss how the technologies outlined here can be adapted for the purposes of VPH-Share and whether they need to be extended given the Project's requirements. The conclusions drawn from this analysis will serve as a starting point for the initial iteration of the WP2 technical architecture, which will be the focus of Deliverable 2.2, due by Project Month 6.

4 Analysis of the State of the Art and Recommendations for VPH-Share

4.1 Cloud application deployment and execution

In this section we plan to describe results of analysis of various technologies and services related to executing applications in the cloud. The scope of our research was very broad, including aspects of virtualization that practically always back the cloud stacks, the stacks themselves, various public cloud providers and, finally, aspects of hybrid clouds, federations and APIs accompanied by libraries enabling manageable use of those APIs.

4.1.1 Virtualization

In theory, cloud computing infrastructures could be built without any virtualization at all, since there exist numerous methods for out-of-band management of physical machines (including control over boot sequences). This includes the so-called Baseboard Management Controller (BMC) integrated into servers (on the motherboard or as an add-on PCI card) such as HP iLO [4], Dell DRAC [5], IBM RSA [6], Oracle (formerly Sun) ILOM [7], Intel RMM [8] or Supermicro Intelligent Management (Nuvoton WPCM450 Controller) [9], in addition to external devices such as LAN/WAN controlled Switched PDUs. Those devices support proprietary command-line interfaces over telnet or ssh and some of them (BMCs) also provide standard protocols such as IPMI [10]. Thus, they can be combined with technologies allowing remote (network) booting of an OS (such as PXE [11][12]) and mechanisms for provisioning mass storage resources via networks, either NAS (e.g. NFS) or SAN (e.g. FC or iSCSI), to create infrastructures with automatic provisioning of full physical (non-virtualized) nodes exposing custom OSs to users. However, complete lack of virtualization would introduce serious problems; specifically:

- significantly increased resource granularity (except storage, which could be provided as needed, over a network) – generally, the smallest CPU/memory unit would be equal to the entirety of resources provided by the least powerful machine;
- increased number of physical machines, as each user would require a full node (and not e.g. 1 core out of many provided by modern servers) – this would greatly exacerbate the storage, power and cooling requirements, thereby increasing operating costs;
- in order to provide various types of machines, the provider would need to maintain various types of physical nodes (e.g. single/dual/quad core processors etc.), which would create a highly heterogeneous infrastructure, very difficult to maintain;
- the infrastructure required to control those nodes would become much more complex and, as such, vulnerable to misconfigurations and failures.

Thus, although virtualization is not required *per se* in a cloud infrastructure, it remains important and allows flexibility which is an essential aspect of modern cloud platforms.

There are various virtualization mechanisms. At the most basic level, we could divide them into two groups – Virtual Machine Manager (VMM, Hypervisor)-based and Operating System-based (also called containers). Hypervisors allow running full operating systems, including separate kernels and user space code, within each VM. As such, they provide better separation and more robust features, including the ability to run custom kernels or kernel modules. On the other hand, container-based VMs run separate user-space code and share common kernels with host OSs. As a result, such VMs have limited features, especially kernel-related (such as iptables in Linux), and lack support for custom kernel modules. They usually won't run OSs which differ from the core OS (perhaps allowing several variants – akin to Linux distributions). There are exceptions, such as Branded Solaris Zones [13], which allow running Linux inside a Solaris Zone – but this is achieved through emulation of Linux system calls by the Solaris kernel, and not by running a true Linux OS. Despite those drawbacks, OS-level virtualization also has some merits – for instance, reduced overhead on host and guest machines. However, as this overhead is not particularly significant on VM-based platforms, it is usually not considered a critical issue. For this reason OS-level virtualization is not commonly applied in clouds – as confirmed by our analysis of the available cloud stacks (described later on in this section), most stacks officially support hypervisor-based technologies, such as Xen [14] or KVM [15]. (Of note is the fact that OpenStack seems to support a single container-based virtualization solution, namely LXC [16] and there is ongoing effort to provide support for OpenVZ [17]).

Hypervisors can be augmented by custom hardware virtualization extensions. The most important of them (VT-x for Intel [18], AMD-V for AMD [19]) enable running VMM in a special ring of the x86 protected mode (called Ring -1 or Root Ring 0), while guest OSs reside in the standard Ring 0 (Non-Root Ring 0), as shown in Fig 1. Without this virtualized guest, the OS kernel would need to run in a less privileged mode (Ring 1 or 3) which would prevent it from accessing restricted CPU instructions and restricted memory regions. Coping with such issues, e.g. in VMWare Workstations [20], involves rewriting instructions and shadowing parts of memory on the VMM level, while in Xen it relies on the so-called paravirtualization (modifying the kernel of the guest VM so it doesn't require privileged access). Both techniques work, but have important drawbacks – including greater overhead and limited ability to run unmodified OSs.

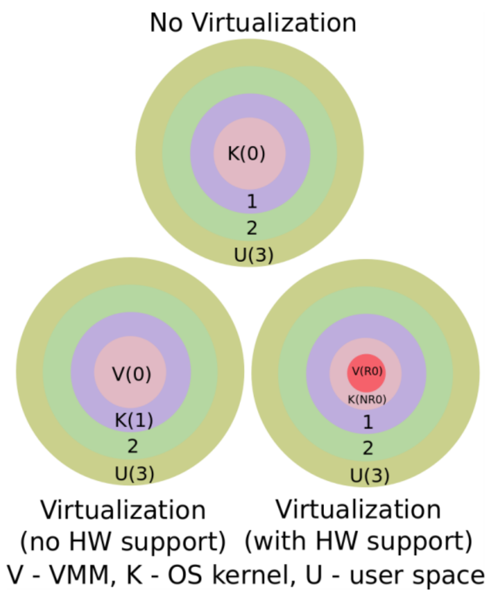
No Virtualization
K(0)
1
2
U(3)
V(0)
K(1)
2
U(3)
V(R0)
K(NR0)
1
2
U(3)
Virtualization (no HW support)
Virtualization (with HW support)
V - VMM, K - OS kernel, U - user space

Figure 4: x86 Protected Mode with and without hardware virtualization support

In addition to the already-mentioned mechanisms, Intel and AMD also provide additional extensions which are not crucial but allow better access to hardware and/or reduce overhead – this includes VT-d/AMD-Vi for direct device access (IOMMU virtualization) or VT-c for I/O and network virtualization.

Each type of virtualization is implemented by various software packages. The most notable examples of hypervisor-based solutions include:

- Xen [14] – highly stable technology providing support for both paravirtualized and fully virtualized guests (with hardware support). It is backed by Citrix and used in their Linux distribution dedicated for virtualization (XenServer); however major Linux distributions such as RHEL or Ubuntu are withdrawing official Xen support in favor of KVM as it is fully integrated into the standard Linux kernel. Xen is supported by all cloud stacks which we have analyzed (even though some of them recommend KVM over Xen);
- KVM [15] – built on the top of the QEMU x86 emulator; however CPU emulation is replaced with full hardware-assisted virtualization. The hypervisor is built into the Linux kernel. Paravirtualization is not supported, hence hardware support is required (otherwise CPU emulation would be needed, which is way too slow for production use). KVM is not as mature as Xen in many aspects – e.g. support for more than 16 VCPUs per VM was

added quite recently (as of RHEL 6 the limit is 64 [21]); however as the project has been taken over by Red Hat (a big player in the market) and is currently the basic technology not just for RHEL but for other popular distributions (such as Ubuntu), it should be well supported and extended in the future. KVM is supported by all of the analyzed stacks.

- VMWare ESX/ESXi [22] – a commercial (but with a free version) bare-metal hypervisor frequently used in industrial applications. VMWare ESX/ESXi is selectively supported by some cloud stacks (such as OpenStack). It comes with a very restrictive default license, disallowing provision of computational services to third parties without special agreements.
- MS Hyper-V [23] – Hypervisor based on MS Windows 2008 Server, supporting various versions of Windows and some Linux releases (SUSE/RHEL are officially supported) [24] as guest OSs. Special Linux kernel drivers are released [25] under the GPL v2 license and are integrated with the Linux kernel. Hyper-V supports up to 4 VCPUs per guest [24] and requires a 64-bit CPU with hardware-assisted virtualization (Intel VT/AMD-V).

The most notable OS-level virtualization solutions include:

- OpenVZ [17] – Open-source component of the commercial product called Parallels Virtuozzo Containers. It supports multiple containers with a Linux OS under a Linux host (same or different distribution), providing the required isolation (including network isolation and enforcing limits on containers). There is ongoing work to fully support OpenVZ under OpenStack [26] (partial support via libvirt is already available). OpenVZ requires a custom-patched Linux kernel.
- Linux-VServer [27] – another solution similar to OpenVZ, also requiring a customized kernel. Less popular than OpenVZ.
- Linux Containers (LXC) [16] – a purely user-space solution (using existing kernel mechanisms so no patching is required – however, the kernel may need to be reconfigured). Supports creating containers. LXC is supported by OpenStack [28].
- FreeBSD Jails [29] – a mechanism providing container-based isolation for the FreeBSD OS, capable of running another instance of the FreeBSD userland (attempts have also been made to run other OSs, such as Debian Linux under the FreeBSD kernel [30]). This technology was originally intended as an isolation mechanism and not a “pureblood” virtualization solution, so it doesn’t provide all the features of the previously-described systems.
- Solaris Containers (Zones) [13] – a similar isolation mechanism for the Solaris OS. In addition to running Solaris guests, it is capable of running certain Linux distributions (so-called branded zones) though Linux system call emulation.

4.1.2 Cloud stacks

In the course of our search for a suitable technology allowing in-house (private) cloud deployment we have studied four software stacks – Eucalyptus [31], OpenNebula [32], OpenStack [26] – including its computational (Nova) and storage (Swift) components – and Nimbus [33]. In the following subsection we present the results of this analysis.

4.1.2.1 Eucalyptus

Eucalyptus is by far the most popular cloud stack in use today. It doesn't require any specific Linux distribution and works well on common releases including CentOS 5.x and Ubuntu 10.x (as Ubuntu Enterprise Cloud). Eucalyptus supports both Xen and KVM so it can be deployed on said systems without the need to manually install a hypervisor which is not well supported in a specific distribution. Eucalyptus provides three types of Amazon-like services/APIs: EC2, S3 and EBS, with good compatibility. Deployment is highly complex as Eucalyptus requires installing numerous services (some may be grouped on a single node for smaller deployments), including: Cloud Controller, Walrus, Cluster Controller, Storage Controller and several Node Controllers. Those services are generally heavyweight. Eucalyptus may operate in one of four network modes: MANAGED, MANAGED-NOVLAN, STATIC and SYSTEM. The first one is the most feature-rich but requires an Ethernet switch configured to allow arbitrarily tagged VLAN (802.1Q) frames to be forwarded. NOVLAN requires a dedicated physical network without any DHCP server running on it. STATIC also prevents external DHCP servers from operating in the network (at the very least, the server must be configured to ignore Eucalyptus VMs) and precludes static mapping of MAC addresses to IPs since IPs are assigned by the Eucalyptus DHCP server. This mode provides much fewer features than the managed modes. Finally, SYSTEM provides the most basic set of features but is also the least complicated as it can rely on any DHCP server already present in the network.

We have tested this stack on a 16-node cluster with the following specifications: Westmere EP 2.93GHz, 48GB 1333MHz DDR3 RAM, 1 x Hitachi HUS154530VLS300 300GB 15000rpm SAS drive using CentOS 5.5 with Xen 3.1.2. Our tests confirm stability and production readiness of Eucalyptus; however we also ran into several issues. The most important are:

- poor efficiency of Walrus and, generally, of VM deployment;
- the SYSTEM networking mode has some glitches which could significantly slow down OS configuration (even though the VM is ready, Eucalyptus does not acquire information about its IP).

However, despite its glitches, Eucalyptus appears to be a reasonably stable solution, useful for larger private clouds (it could be too heavy for smaller deployments).

4.1.2.2 OpenNebula

OpenNebula is a lightweight cloud stack which only requires a small set of scripts written in Ruby set up on the main (controlling) node. No specific scripts need to be installed on Worker Nodes (running the VMs) – all necessary scripts are uploaded and executed when needed by the main node through the ssh protocol. As the stack is not complex, there are no hard requirements related to the operating system; however two specific factors may influence OS selection: the stack runs much more smoothly on KVM than on Xen and it requires shared storage – either an NFS volume (which should be well supported by any distribution) or cluster FS over SAN (e.g. GFS2, supported in RHEL/CentOS over iSCSI). In addition to an inbuilt (XML-RPC) API, OpenNebula provides an OCCI API and a limited

EC2 API. There are three network configuration modes. Two are based on an onevnet mechanism which assigns MAC addresses to the VM generated from the given IPv4 address (the last 32 bits of MAC are equal to the IP). IP is either configured by a small script running inside the VM, extracting the IP address from the prepared MAC address (a type of stateless auto-configuration), or it may be assigned by an external DHCP server. The third network mode relies on contextualization in which each VM runs a specific script from a mounted volume and may use custom methods to assign an IP address.

We have tested this stack on 12 cluster nodes with the following specification: HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420 16 GB RAM 667 MHz DDR2 1xHP GJ0120CAGSP 120GB 5400rpm SATA drive using CentOS 5.5 with Xen and KVM and GFS2 over iSCSI (2 TB) as Shared Storage. Our tests confirm that the presented features are generally adequate for a private cloud installation; however we have also identified some problems that need to be taken into account:

- EC2 compatibility is not stable and causes serious problems – it may need to be fixed if such compatibility is required (the OCCI API works as expected);
- we have had problems booting certain VMs under Xen (everything seems to work under KVM);
- there are some minor glitches, including problems with resource discovery on non-US locales (e.g. Polish hosts).

In our opinion, OpenNebula appears to be a good choice for small private cloud deployments, especially when no complex functionality is required.

4.1.2.3 OpenStack computations (Nova/Glance)

We have analyzed two releases of OpenStack: the next-to-current one (Bexar) and the current one (Cactus). In both cases we’ve analyzed Nova components and, additionally, for the Cactus release we've also tested the Imaging Service (Glance) used to distribute VM images. OpenStack is a large, feature-rich stack targeted for building Compute Clouds (described in this section) and Storage Clouds (described in the following section). Due to some complex dependencies, it’s *de facto* meant to run on Ubuntu 10.04 or later. Another supported (but discouraged) distribution is RHEL6. RHEL/CentOS 5.x are not supported and it is very difficult to set up OpenStack on these OSs due to numerous missing dependencies. OpenStack officially supports multiple hypervisors [28], including KVM, Xen, Hyper-V, WMWare ESX and UML as well as one OS-level virtualization solution – LXC. There is also ongoing work to fully support other solutions such as OpenVZ [34]. In contrast with the other stacks described in this section, there is no strict division between “management” and “worker” nodes. Instead, there is the notion of Cloud Controller and Compute Nodes; however, in a default setup, Cloud Controllers can also run the Nova compute service and, as such, act both as management and worker nodes (running VMs). The CC node is distinct only in the sense that it runs some additional services – both Nova-specific (API, network, scheduler) end external (RDBMS – MySQL or PostgreSQL and RabbitMQ). Those additional services are accessed and used by all nodes to obtain configuration and run VMs. OpenStack provides two mechanisms used to distribute VM images. The legacy mechanism, called Nova ObjectStore (not to be confused with OpenStack ObjectStore – Swift) is a simple service running on computing nodes and storing images in the file system. The new mechanism, called Image Service (Glance) is a registry which provides centralized access to images stored by one of the following backends: local file system, OpenStack Object Storage (Swift), S3 (Amazon or compatible service) or HTTP (read-only mode). Open Stack Nova provides an EC2 API which does not fully support all the features offered by Amazon but provides decent compatibility. There are also three network configuration modes controlled by so-called network managers: VlanManager, FlatDHCPManager and FlatManager. The first one is the default and most feature-rich – it automatically creates a VLAN (or multiple VLANs – one for each subnet) and bridges bound to appropriate VLAN interfaces, and provides IPs through a DHCP mechanism. Of course, this mode has the tallest requirements as it calls for a switch supporting VLANs, configured in such a way that it allows OpenStack VLAN-tagged frames to pass through. However, only predefined VLAN tags need to be configured as tag IDs may be assigned by the OpenStack DB. The second mode also uses DHCP to provide IP addresses but it does not automatically create VLANs – instead, the cloud administrator has to provide a separate unused network interface (with connectivity), either physical or virtual (e.g. a manually created VLAN). The last mode has the least requirements: it can share a single network and does not conflict with other DHCP servers (because it does not use any); however it requires manual configuration of appropriate network bridges by the administrator and restricts guest VMs to those using specific Debian-like network configuration (in this mode OpenStack injects static network configuration into the VM image).

As already mentioned, we have tested two releases of OpenStack Nova – the Bexar release was tested on a 13-node cluster with the following specification: HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420 16 GB RAM 667 MHz DDR2 1xHP GJ0120CAGSP 120GB 5400rpm SATA, while the Cactus release was tested on 10 SuperMicro nodes with 4 x Intel Xeon E7-8867L 128 GB RAM 1333 MHz DDR3 ST3450857SS 450 GB 15000rpm SAS HDD. Both systems ran under Ubuntu 10.04 LTS with KVM. We tested all available network modes. During tests the following issues were observed:

- VMs have problems getting network configuration when using FlatManager. This is related to problems with injecting appropriate configurations (however, VLANManager works seamlessly).
- Nova schedulers have some issues – the default (Zone) scheduler deploys VMs at random and overloads some nodes while leaving others practically idle. The second standard scheduler (Simple) works well (following a small patch which increases the allowed number of cores per node) but it doesn't offer advanced features. Note that developers are working on more advanced schedulers such as Heterogeneous Architecture Scheduler [35] and Policy & Constraint based Scheduler [36].
- On the second cluster (with 40 cores per node) we were able to hit the KVM VCPU limit per VM (16 using the presented software stack); however, it seems that since RHEL6 this limit was raised to 64 VCPUs (also in the stock kernel) so it should be possible to use the KVM version for OpenStack
- When trying to boot a large number of VMs at once (above 50), the Cloud Controller node is overloaded by the Nova-API service and some VMs fail to boot. Optimization of

this service or a load balancing strategy may be needed if a large number of VMs are to be instantiated concurrently.

In spite of those limitations, most likely caused by the relative immaturity, OpenStack remains a very feature-rich and promising cloud platform. After resolving some issues it might be a good candidate for large and medium-sized private cloud systems as it strikes a good balance between functionality and system requirements.

4.1.2.4 OpenStack storage (Swift)

As already mentioned, OpenStack also supports a Storage Cloud Solution called Swift. This solution is used, e.g. by the RackSpace public cloud provider, as the basis of their storage service called Cloud Files [37]. Its architecture is based on two management services called Proxy and Auth Service which need to be deployed on a separate node (or nodes) connected to both public and private network as well as several Storage Nodes in the private network. Storage Nodes are grouped into so-called Zones – each need to contain at least one node. To provide an adequate level of redundancy there should be at least 5 zones and they should use separate network/power subsystems [38]. Storage nodes need to use highly efficient hard drives and network connectivity. RAID is not recommended as redundancy is provided by the stack itself and RAID would just slow down write operations (according to developers) [39]. The stack provides a dedicated RackSpace CloudFiles API. There is also work on an S3-compatible API [40] but, as of this deliverable's submission date, S3 support remains highly experimental.

We have tested the stack on 16 nodes (1 dedicated Proxy/Auth node and 15 Storage nodes grouped into 5 zones) – each with the following specification: HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420 16 GB RAM 667 MHz DDR2 1xHP GJ0120CAGSP 120GB 5400rpm SATA. As expected of a production-grade stack used by large cloud IaaS providers, we've experienced few problems with the native RackSpace CloudFiles API.

4.1.2.5 Nimbus

Nimbus is a relatively small and lightweight stack based partially on the Globus Toolkit. Its architecture consists of a dedicated Service Node, running more complex parts of the software stack and responsible for API and scheduling VMs, and VMM nodes, running much lighter software responsible for operating the VMs. There are no strict requirements regarding the OS: Nimbus may use either Xen or KVM. Xen seems to be better supported but, as we have verified, KVM also works. The stack may use standard schedulers such as PBS to schedule VM execution. Communication between services is done using the ssh protocol. Images may be provided either by a dedicated storage service called Cumulus, installed on a Service Node, or by a LANTorrent mechanism. There are two network configuration mechanisms: Local, running a DHCP server at each VMM node, and Global, generating DHCP configuration files that could be integrated with an existing (auxiliary) DHCP server running in the network. Nimbus offers several APIs: a WSRF-based native API, Amazon EC2 WSDL and Query API for the computational features, and an Amazon S3 API for Cumulus.

We have tested the stack on 9 nodes (1 Service and 8 VMM) with the following specification: HP ProLiant BL2x220c G5, 2 x Intel Xeon L5420 16 GB RAM 667 MHz DDR2 1xHP GJ0120CAGSP 120GB 5400rpm SATA running Ubuntu 10.04 LTS and KVM. During our tests we ran into serious issues with EC2 API accessibility. In light of these problems we did not perform in-depth tests, as, in our opinion, the three stacks described in the previous sections are better suited for use in any type of private cloud deployment.

4.1.3 Public cloud services

There are numerous commercial entities providing services using the cloud model [41]. Such services could roughly be classified into three groups, as shown in Figure 5.

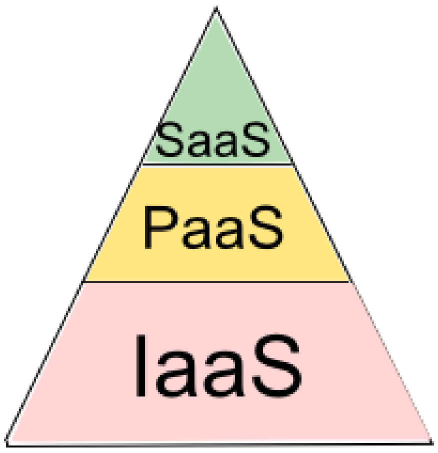

Figure 5: Classification of cloud services

Infrastructure as a Service (IaaS) is the most basic type of service, which enables the client to run custom software, including a custom operating system. Platform as a Service (PaaS) provides a much higher level of abstraction and, as such, simplifies service usage but imposes some restrictions on the user such as the need to apply technologies supported by the platform (however, users can still develop their own solutions based on the provided platform). Finally, in the Software as a Service (SaaS) model, the user is provided with a special-purpose application. This model does not require any knowledge related to software development as the user is simply using the provided software, although – naturally – it does restrict the user to features provided by the software. The aforementioned diversity of public cloud solutions makes it hard to choose the appropriate platform, even though there are some criteria which should be taken into account, including cost, efficiency and geographical location. As a result, some solutions such as CloudCmp [42] were developed to assist the user in making this decision. Below we present some notable examples of various types of cloud systems, both commercial and scientific.

4.1.3.1 Commercial public cloud providers

Commercial cloud providers offer services to all potential users willing to pay for them (barring violations of the agreed-upon terms of service). This allows any entity to acquire a relatively large amount of resources when needed without the need to buy and maintain large IT infrastructures. Of course commercial clouds can also be used by scientific projects providing those projects have sufficient funds for it or obtain scientific grants as is the case with VPH-Share [43].

Notable commercial cloud providers include:

- Amazon Web Services [44] – currently the market leader in IaaS provision, offering multiple cloud services, particularly the Elastic Compute Cloud (EC2) which supports arbitrary user VMs in the cloud, and the Simple Storage Service (S3) for storing large quantities of data. AWS also offers many additional services, such as:
  - Computations: Elastic MapReduce – supports computations using Apache Hadoop running on EC2 resources, with Auto Scaling – allowing up- and downscaling EC2 instances as required; Both services are provided at no additional cost.
  - Storage: Elastic Block Storage (EBS), i.e. persistent volumes which may be mounted and used as raw storage devices by EC2 instances – this facilitates creating a custom file system which is preserved when the instance shuts down. EBS may also be used as a boot device for EC2. The user is charged for the volume of the stored data and the number of I/O requests. The AWS Import/Export service allows storing or obtaining large quantities of data to/from S3 by sending physical devices via traditional mail (internal HDD or external device with eSATA or USB interface). The cost of this service is calculated as the sum of a flat fee for each device and the time needed to upload data.
  - Messaging: Simple Queue Service (SQS) enabling creation of queues for sending simple messages, Simple Notification Service (SNS) – supports topic-based HTTP notification; Simple Email Service (SES) – enables distribution of large quantities of e-mail messages (such as newsletters).
  - Content Delivery: CloudFront – allows serving content originally stored on S3, from various locations that are nearest to the user, to reduce the latency and maximize data rates.
  - Monitoring: CloudWatch – may be used to monitor EC2 instances.
  - Database: SimpleDB – provides a scalable non-relational data store; Relational Database Service (RDS) – provides a highly scalable access to MySQL RDBMS (support for Oracle 11g is planned) with the ability to run a separate standby instance in another part of the world (i.e. in a separate Availability Zone).
- Rackspace [45] – another popular IaaS provider, offering three types of services, including a computational service called CloudServers which can run custom cloud instances (like Amazon EC2). In contrast to Amazon, instances have dedicated storage and as such are persistent. The RackSpace storage service called CloudFiles enables storage of arbitrary data in the cloud (much like Amazon S3), while Cloud Load Balancers provide a load balancing mechanism for various protocols such as HTTP(s), LDAP(s), FTP, IMAP, POP3(s) and SMTP. For all services, RackSpace uses its custom RESTful API. Rackspace is currently applying the previously described OpenStack.
- Google – offers a PaaS solution called Google App Engine [46], which allows running software created using a vendor-supplied SDK that is available for Java (version 6) and Python (version 2.5.2). Google also offers various SaaS services such as GoogleDocs, Gmail, Google Calendar etc.

- Microsoft – offers a PaaS solution called MS Azure [47], composed of Windows Azure and SQL Azure. The former allows running custom applications created for the .NET platform as a Web application (Web role) or a standalone tool (Worker role). It is also possible to run Windows Server 2008 R2 Virtual Machines from customized VHD images using the so-called Virtual Machine role. This mode of Windows Azure could be categorized as IaaS. The latter component, SQL Azure, supports cloud-based relational databases and should also be considered IaaS. In addition to Azure, Microsoft offer pools of SaaS solutions called Microsoft Online Services – including such services as Exchange Online, SharePoint Online or Office 365 (beta version).
- Salesforce [48] – offers various cloud-based solutions dedicated to the business community, including Force.com. PaaS enables creation of custom business applications (using a graphical editor) and provides a SaaS CRM system called Sales Cloud.
- Heroku [49] is a Ruby PaaS solution. It provides a simple deployment mechanism based on Git or a dedicated client. The deployed application is run in the cloud as a dedicated process (called “dyno”). It executes in the context of a dedicated user account and provides all the required elements of the Ruby software stack such as the Ruby VM, application server, web service interfaces and middleware. External users access such applications through a multi-layer infrastructure consisting of a HTTP reverse proxy, HTTP cache and the so-called “routing mesh”, responsible for load balancing between dynos. Ruby apps may use various solutions (such as Rails and Sinatra), and use a user-supplied RDBMS – either PostgreSQL (the default one) or another system provided as an add-on (Amazon RDS/MySQL, MongoDB, Redis or CouchDB).
- PiCloud [50] – a Python-based PaaS with a minimal API comprising a single dedicated library. It can run arbitrary Python functions in the cloud (with the simple cloud.call() function) as well as control the status of executed functions and retrieve results once a given task is completed. There are also additional features such as data storage, all encapsulated by the presented library.
- GoGrid [51] is an IaaS provider offering several types of services: Cloud Servers – typical computational cloud solution running OS images in a virtualized environment; Dedicated Servers – permanent physical machines billed in monthly/yearly cycles; Load Balancers – assigning specific IPs to be balanced between servers. Users may choose to use Round Robin or Least Connect (send connection to the server with the lowest load) and, additionally, select a persistence option to trace SSL sessions or connections from specific source IP addresses and redirect them to the same server (e.g. to prevent sessions from breaking down). Another standard service is Cloud Storage, which can be accessed with standard protocols such as SCP, FTP, SAMBA/CIFS or RSYNC. Each service can be controlled through a web interface, iPhone Client or dedicated GoGrid REST API. Additionally, some advanced features are provided, including the Content Delivery Network which can acquire data from the location nearest to the user (equivalent to Amazon’s CloudFront) and dedicated hardware firewalls (Fortinet by default; Cisco ASA 5510 on special request).

4.1.3.2 Scientific public cloud providers

Despite the fact that the cloud market is largely dominated by commercial service providers, some clouds are dedicated purely to scientific research. An example of this approach is the group of small clouds described by the University of Chicago on the "Science Clouds" website [52], including Nimbus at the University of Chicago, Stratus at the University of Florida, Wispy at Purdue University, Kupa at Masaryk University.

Another example of a much larger scientific cloud platform is the set of cloud services provided to scientists by the FutureGrid Project [53]. It offers access to various cloud platforms such as Eucalyptus, Open Nebula and Nimbus and allows conducting computational experiments on the infrastructure it controls.

NASA Nebula Cloud (http://nebula.nasa.gov) is a cloud computing infrastructure deployed at the NASA Ames Research Center. It is housed in a container-based datacenter and is developed using open-source software, including the OpenStack system. The goal is to provide compatibility with Amazon EC2 APIs and provide a facility through which to share NASA resources and data with the public. The Nebula project is one of the contributors to the OpenStack platform.

SARA (a Dutch high-performance computing center) has launched an HPC cloud environment for evaluation purposes. This infrastructure (which can be found at http://www.cloud.sara.nl/) bases on OpenNebula, although SARA has developed a cloud management console, subsequently contributed to the OpenNebula project.They seek applications which cannot be run on existing HPC, cluster and Grid infrastructures – including e.g. laptop cloning, scientific code which requires custom libraries, large database processing, etc. Beta users are invited to evaluate this installation.

CC1 is a project (2009-13) run by the Institute of Nuclear Physics, Polish Academy of Sciences in Krakow. The goal is to build a private cloud and make it publicly available for scientists as well as commercial startups. CC1 develops its own cloud management software based on OpenNebula and intends to tackle many low-level issues, including the provisioning of private networks and user-friendly Web tools.

The Open Cloud Consortium (http://opencloudconsortium.org/) is an initiative to build an open cloud infrastructire for researchers, focusing mainly on bioinformatics, astronomy and data mining applications. The participants are required to contribute a container-based datacenter and 100GB/s network links. The Open Science Data Cloud supports e.g. BioNimbus and Sloan Digital Sky Survey projects.

4.1.4 Hybrid clouds

Hybrid clouds are based on connecting in-house (private) clouds with public clouds. This allows exercising tight control over critical data (processed in-house) while allowing some less confidential data to be processed in the public realm. Although hybrid clouds are not yet very popular [54], some solutions have specifically been developed to support them. OpenNebula and Nimbus provide special EC2 drivers which allow deploying images in the Amazon public cloud. This makes it possible to create hybrid clouds, bridging local installations and Amazon. On the service providers' side, there are such initiatives as the Amazon Virtual Private Cloud (Amazon VPC) [55] (still in beta) which can isolate sections of AWS resources and connect them with regular private clouds using VPN. GoGrid also offers a hybrid solution, based on using the vendor-provided Dedicated Server as the “private” part and additional Cloud Instances as the “public” part of the infrastructure (when more resources are needed, or for less critical data processing).

4.1.5 Cloud federations

Cloud federations work by extending the concept of a hybrid cloud in such a way that not only are public and private clouds integrated, but computations can be freely moved between multiple clouds of both types. The task is very complex since various providers use their own, incompatible APIs (as discussed in previous sections). In spite of such problems, it is clear that federated solutions are important [56]. At present there are still no generic and common standards in this area – just some partial solutions like those described in the above section, with some standardization efforts related to establishing common image formats (e.g. OVF); however work is ongoing and ambitious plans such as live migration between clouds are being made [57]. It should be noted that VPH-Share is an attempt at addressing this issue in the scope of medical data management and application deployment.

4.1.6 Cloud APIs and libraries

As already mentioned, there are multiple nonstandard APIs used by various public cloud providers. Despite being similar (mostly REST-based), they are not compatible. Cloud stacks also come with their own APIs; however – fortunately – most of them support EC2/S3 APIs, which, despite being custom (developed by Amazon), have become a *de facto* standard in the area of cloud computing. In addition to those APIs, standardization efforts have led to the creation of the Open Cloud Computing Interface [58] by the Open Grid Forum. This interface is already supported by e.g. OpenNebula, while other cloud stacks, as well as some service providers, are also working on implementation of OCCI potentially paving the way for future API standardization. Another way to provide an interoperable API is suggested by the authors of Deltacloud [59]. Instead of imposing a custom API on service providers and cloud stack developers, they have decided to develop a tool providing its own REST API capable of interfacing with multiple cloud APIs already offered by various providers. The project also offers a dedicated Ruby client.

As most APIs are REST-based, they can easily be invoked from most languages without additional libraries. However it’s definitely simpler to use a joint, standardized API. Some providers such as Amazon offer a dedicated cloud SDK (currently for Java, .NET, PHP and mobile devices – Android and iOS). AWS APIs bindings for other languages are supported by third-party libraries such as the Ruby library provided by RightScale (right_aws) [60]. On the same site there are also additional libraries for other providers, including GoGrid.

4.2 Cloud resource allocation management

Cloud computing [61], [62] provides indisputable advantages for its users. It is perceived as an environment with a theoretically infinite capacity where resources are provisioned on demand and the user is charged on a pay-per-use basis. This makes the computing infrastructure very elastic and scalable, while reducing the service provider’s operating costs [63]. Nevertheless, cloud computing also carries many threats due to the fact that the actual computation is performed on a third-party physical infrastructure which is beyond the control of the end user. The issues associated with this mode of operation include compromised or leaked data [64], [65], or poor reliability of the infrastructure itself [66].

Within the VPH-Share project Atomic Services will be deployed in a cloud environment, including private and public clouds. The services will access and process sensitive medical data. One of the main WP2 challenges is therefore to deploy Atomic Services in a way that ensures optimal performance, is cost-effective and preserves security constraints. Applications in the cloud can be scaled both horizontally (by choosing the capabilities of the virtual machine(s) on which they are deployed) and vertically (by instantiating many virtual machines for each Atomic Service). Making such decisions requires the platform to model application performance and resource assignment and therefore calls for a heuristic resource allocation manager. The majority of existing managers implement an algorithm involving:

- estimating resource demands based on online and/or historical performance monitoring;
- optimizing the allocation according to given criteria;
- choosing resource assignments.

This chapter gives an overview of the state of the art in cloud resource allocation. It identifies scientific issues associated with the first two steps and studies potential approaches to solving those problems. The final step of the presented algorithm is covered in Section 4.2.3.

4.2.1 Deploying an application in the cloud

One of the most important challenges we face in the scope of VPH-Share project is to provide a simple way of migrating existing applications, which are typically standalone executables, into the cloud environment, as outlined in Section 2.1. This process need to fulfil the following requirements:

- support for wrapping existing command-line applications with minimal developer involvement;
- the application needs to expose a remote interface – in the scope of the Project we will use SOAP [67] or RESTful [68] Web services;
- the application needs to deal with a dynamic environment (IP address, amount of RAM, etc. can be changed);
- the wrapper has to be started whenever the underlying Virtual Machine is turned on.

As this problem is not new, or indeed specific to VPH-Share, there exist numerous tools and frameworks which support similar scenarios. SoapLab [69] is an example of such a tool. It allows developers to dynamically generate and deploy a web service on top of existing command-line applications. It delivers two solutions for generating web services. The first one creates a generic web service, where parameters are passed using a key-value schema. The second one is much more advanced – it can generate a web service with a custom interface. All the steps involved in generating web service can be performed without in-depth knowledge of computer science. The Generic Service Toolkit [70] is another example of a command-line tool wrapper. It delivers a framework for configuring the requisite infrastructure for remotely run command-line applications, starting with a definition of a machine farm where the command-line application will be executed, through wrapping command-line tools as web services, and finally providing a portal which allows users to invoke the wrapped applications (see Figure 6).

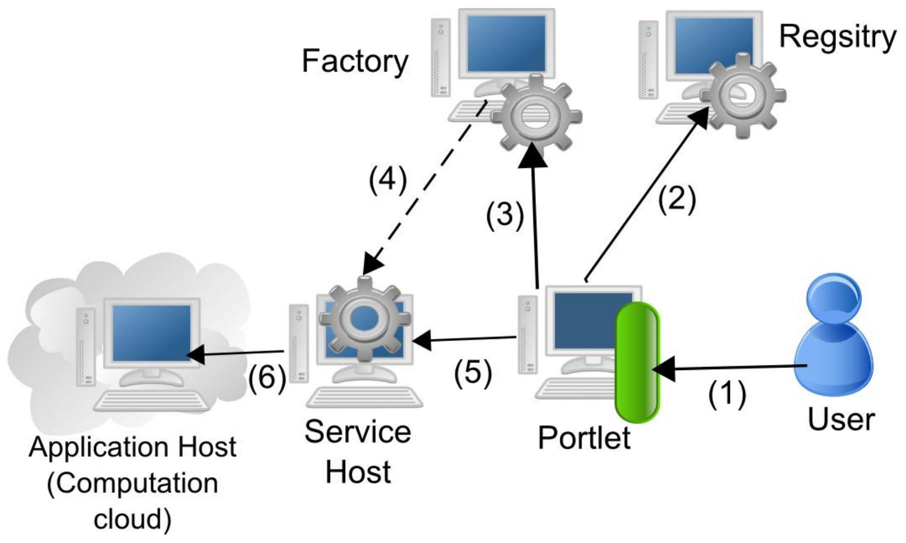

Figure 6: Generic Service Toolkit architecture

(source: http://www.extreme.indiana.edu/gfac/userguide.html)

The presented solutions are perfect for applications which are functionally complete – i.e. do not require any processing of the parameters sent to the web service or REST. For example, if a command-line tool processes large files, sending such files using SOAP might not be the best idea. An even more complicated situation may occur in a cloud environment, where dedicated cloud storage is used to host large files. In such cases, the optimal solution would be to send file location information to the web service and add additional logic to download files, assign them with unique names and pass their location to the command-line tool. A reverse procedure could be applied to files produced by the application. In such a scenario, the person responsible for wrapping command-line tools would need to implement additional web service logic. To simplify this process two types of tools can be applied:

- command-line wrapping libraries;
- Web/REST service libraries.

Nowadays, most programming languages provide a library, which simplifies the process of wrapping and using command-line applications with a dedicated API. An example, written in Java, is the Apache Commons Exec library [71]. The situation is even more straightforward in dynamic languages like Ruby [72] or Python [73], because developer-friendly libraries are included in the SDK. If a command-line application is to be made accessible via an API, its features have to be exposed remotely. For this purpose one of the available frameworks for creating Web or REST services can be used. Java SDK in version 1.6 has built-in support for Web and REST services, but numerous advanced libraries also exist (which implement a greater portion of the WS-* specification). One of the most well-known tools is CXF [74]. In Ruby, soap4r can be used, while in Python pywebsvcs [75] fulfils a similar function.

4.2.2 Estimating resource demands

In order to estimate resource demands of an Atomic Service and prepare an optimal deployment plan, it is necessary to study the characteristics of the service in question. This includes:

- resource demands, such as CPU, memory and storage;
- input and output data transfer through the network;
- well-defined interaction model.

While the first two aspects are self-explanatory, the last one calls for a more detailed explanation. Although an Atomic Service exposes a HTTP-based interface it may not employ a request-response interaction model. It is possible for a single request to involve several processing steps – for instance submitting a request and obtaining a request ID, polling for request status and retrieving results. Therefore the duration of a single HTTP request does not provide meaningful data for performance analysis.

Regardless of whether performance data is used in an offline or online manner, it must be collected and resource consumption must be associated with each request. [76] suggests an interesting approach that relies on black-box application monitoring and correlation of system load with application logs. The idea is to investigate machine-level load indicators such as CPU or memory usage. This approach has many advantages from the VPH-Share perspective:

- it is universal and can be applied to arbitrarily complex Atomic Services;
- it is unintrusive and doesn't affect running Atomic Services; furthermore, it doesn't require any additional effort on the part of service developers;
- software components required by the Atomic Service are not instrumented so they can be easily updated or replaced;
- the Atomic Service can use legacy or third-party components which don't provide source code.

Assuming that system load is generated only by serving requests (i.e. there is only one Atomic Service installed on a machine), one can implement the following stepwise algorithm to discover how serving requests affects the load of the server:

- split server logs into *n* intervals;
- assign *m* request types for each interval;
- represent the number of request of type *j* in interval *i* as an element of an *n* by *m* matrix;
- construct a vector representing the load for each system resource;
- compute the correlation between columns of the matrix and the given vector.

A completely different approach is to insert probes into the software and collect low-level information on application performance [77]. According to [78], instrumentation techniques can be categorized as:

- source code instrumentation [79] that modifies the source code of the application;

- binary wrapping [80] that replaces library functions with wrappers invoking original functions;
- binary editing [81];
- runtime patching that modifies the code of a running process in system memory [82].

4.2.3 Optimizing resource allocation

When discussing dynamic resource allocation on the cloud for VPH-Share workflows, there is a need to define the optimization problem in terms of goals (or criteria) which will be optimized. These criteria can be classified as quantitative properties of the solution, as well as some quantitative constraints that need to be fulfilled. As there are often multiple conflicting criteria which cannot be simultaneously optimized, there is a need for multicriterial optimization strategies which take into account the necessary trade-offs.

The first performance criterion to minimize is time. This usually includes the sum of computing time and resource acquisition overhead, resulting in task completion time which is the most important metric from the point of view of end users. In the case of public clouds, the resource acquisition overhead is on the order of 1-5 minutes [83], so it can be considered negligible for long-running services. Similar overheads can be achieved on private clouds running Eucalyptus, OpenStack or other systems described in Section 4.1.2. On the other hand, in the case of private clouds with a limited amount of resources, resource acquisition may involve longer queues and the problem becomes similar to batch scheduling in clusters and grid systems.

The second most widely used criterion for optimization is cost, which obviously results from the Amazon pay-per-use pricing model. The actual cost depends on the application performance characteristics: for CPU-intensive applications the CPU-hour cost dominates, while for data-intensive workloads the cost of data transfers becomes the most important [84]. Armbrust *et al.* [63] notice that the price of transferring data decreases more slowly than the price of computation. Another issue comes up in the context of parallel workloads: usually the parallel efficiency of programs run on cloud resources is not close to 100%, which means that adding computing nodes to an application yields improvements which are not directly proportional to cost increases. This is an example of a trade-off which has to be taken into account.

Arguably the third most important criterion for resource allocation optimization is resource reliability. Public clouds typically offer some form of SLA which is usually fulfilled, barring infrequent (but inevitable) service outages. On the other hand, the resources offered by private clouds or research projects such as FutureGrid (futuregrid.org) are free but often highly unreliable. Some tasks in the workflow may need to be allocated to more reliable resources, while others – such as parameter sweep workloads – may be successfully executed on less reliable ones.

Finally, there are some qualitative requirements, such as minimum amount of memory or disk space availability; and quantitative ones, such as access to locally available datasets (moving computations to data) or security perimeter limitations.

4.2.4 Cost model for clouds

One of the main advantages of cloud computing is that it introduces a business and cost model where computing power is billed per CPU-hour usage. This is an important change of perspective and allows treating computing as a commodity. For business operations this means a shift from capital expenditure to operational expenditure, which is usually convenient and economically desirable. Similar budget considerations apply to research projects where grant funding is required to buy and operate computing hardware, while adopting a pay-per-use model would be more economical.

However, it turns out that fully understanding the cost model of the cloud is not a trivial task and, in fact, becomes an additional complex problem which needs to be solved. The following issues have to be taken into account:

- multiple cloud providers (Amazon EC2, RackSpace, etc.) with varying pricing models;
- multiple instance types with different performance characteristics;
- hourly billing cycle;
- data transfer costs for inbound and outbound network traffic;
- cloud storage costs and related data transfer rates;
- discounts for using “local” storage (e.g. there are no transfer costs between Amazon S3 and EC2).

The performance of different cloud instance types can be assessed using different benchmarks. Examples include scientific benchmarks, such as the NAS parallel benchmark suite [85]. Benchmarks for typical business applications can be found on the CloudHarmony portal [86]. They try to normalize the performance of different instance types using a unit called CCU (similar to the Amazon-introduced EC2 Compute Unit (ECU), which is equivalent to 1.0-1.2 GHz 2007 Xeon performance).

One important aspect is the market model offered by “spot instances” at Amazon, which provides the opportunity to buy instances based on availability. Historical data can be found e.g. at [87].

4.2.5 Sample solutions, models and techniques for resource allocation management

Multicriterial optimization usually involves finding a set of Pareto-optimal solutions, i.e. eliminating all solutions dominated by others, which are better with respect to one criterion without sacrificing any of the others. From these Pareto-optimal solutions, an arbitrary decision has to be taken to select the final solution, either taking into account a trade-off or some aggregation function (e.g. a weighted sum) over the relevant criteria. Such decisions can be taken either directly by users, or by systems based on a defined policy. An example of using Pareto-optimization is discussed in [88]. This publication presents a system responsible for provisioning resources in a cloud-like environment in such way as to minimize costs and maximize performance. As these criteria are often found to be in conflict, the authors propose a Multi-Objective Genetic Algorithm (MOGA) [89] to produce the approximation of a

Pareto-optimal set of solutions and then apply a normalized objective function to select one solution.

A very promising nonlinear programming approach to minimization of execution cost is presented by Pandey *et al.* [90]. The authors focus on cost analysis of operating an intrusion detection workflow. They denote the workflow as a Directed Acyclic Graph in which nodes represent computational tasks while edges correspond to data dependencies. Their model includes a set of storage sites, a set of computing sites and a set of tasks. It also assumes that the cost of computation for a known size of input data is known. This cost becomes inversely proportional to computation time while the cost of data transfers is fixed by providers. In this way, the total cost of executing the workflow can be defined and subsequently minimized by finding a feasible set of datasets that need to be transferred from storage to computing sites such that the total transfer and computation cost is minimal and all data dependencies are preserved. The model is expressed using the Modelling Language for Mathematical Programming (AMPL) [91] and solved using DONLP2 [92].

Chandra *et al.* [93] proposes a generalized processor-sharing server [94] as an abstraction of resources and represents application load as a system of queued requests. For every application there is a maximum acceptable time within which the request must be served. Optimization is based on minimization of a utility function which grows each time the maximum response time is exceeded.

A similar approach is presented by Van *et al.* [95] who investigate optimal planning and virtual resource management for clouds preserving SLA contracts. Optimal placement is based on utility functions and a Constraint Satisfaction Problem (CSP) solved with the CHOCO tool.

The EC-funded FP7 RESERVOIR project [96] brings significant contribution in the area of exploiting cloud environments. Its main goal is to create an ecosystem where business application can be executed in a federated cloud taking into account the defined criteria. Each cloud provider creates and manages RESERVOIR sites – a physical infrastructure with defined hardware and additional RESERVOIR software installed. A set of RESERVOIR sites creates a RESERVOIR Cloud. By using this federation, the user is able to execute business applications. A Service Applications is a set of software components, which, composed together, solves the stated problem. Every component of the Service Application can be deployed on different a RESERVOIR site. The decision on where the application should be deployed is taken by the Service Manager component, which consumes the Service Application manifest. From the manifest, the Service Manager derives a list of resources, their configurations, pricing policy, etc. for every component of the application. While existing cloud infrastructures are quite inflexible in defining the best deployment plan (they usually take into account only one indicator, such as price, computing power etc.), the Service Manager tends to rely on multiple criteria while deploying applications. Once the specification of the application is retrieved, deployment can be performed: the Service Manager contacts the underlying layer – the Virtual Execution Environment Host (VEE Host) – to process the deployment, based on the data retrieved from RESERVOIR sites (e.g. available hardware and software, performance, SLA, pricing policy) and application requirements. While the application is being deployed, the Service Manager monitors the environment and checks whether the defined high-level/business SLA is fulfilled. If not, the application can be reconfigured (e.g. the number of VEE Hosts can be increased or decreased, or additional memory can be added to existing VMs). The VEE Host is also responsible for guaranteeing service-level agreements on the technical level (by monitoring overload, used CPU, etc.) It can change the configuration of managed VMs, or even migrate some parts of the application to other VEE Hosts.

Another interesting system dealing with cloud management is Haizea [97]. It is a lease manager where the lease is a contract between the user (I want access to resource *X* for the period *Y*) and the provider (who agrees to deliver the required resources). In Haizea resources are understood as the required hardware, software and a reservation period time. Haizea is designed specifically for the cloud, thus it factors in the cloud-related overhead (e.g. image migration time, booting time) when delivering the required service. Owing to integration with OpenNebula (Haizea can replace the OpenNebula manager) it can manage clouds based on Xen, KVM or VMWare.

An interesting idea for minimizing High Throughput Computing costs is demonstrated in [98]. The authors notice that in order to ensure on-demand access to resources, providers must overprovision their infrastructure. They claim that by deploying backfill virtual machines on idle nodes, providers can increase IaaS cloud utilization from 37.5% to 100%.

A significant contribution to resource allocation management is represented by autonomic computing [99]. Autonomic environments feature important self-management properties, i.e. self-configuration, self-healing, self-optimization and self-protection – thus they are also referred to as self-* systems [100]. There are attempts to apply autonomic resource allocation in small-scale data centres [101], [102]. Although mature and advanced techniques such as machine learning [102], queuing models with heuristics [103] and control theory [104] are available, building a fully self-managing system remains a challenge [105].

4.2.6 Cloud allocation simulators

Dynamic resource allocation and distributed resource management on grids and clouds raise many challenges due to their inherent complexity. In addition to theoretical studies, the proposed algorithms have to be subjected to experimental evaluation, either in real or simulated environments. Regarding experimental test-beds, projects such as Grid'5000 (https://www.grid5000.fr) and DAS-4 (http://www.cs.vu.nl/das4/) have been deployed in Europe, while FutureGrid (https://portal.futuregrid.org/) is provided in the USA. These projects enable running computer science experiments on dedicated resources using on-demand provisioned middleware or cloud computing stacks. As experimental evaluation of algorithms on real computing platforms is not always feasible, a number of simulation toolkits dedicated to the problem have been developed. Among them the most interesting ones from our perspective are GridSim, CloudSim, GSSIM and GroudSim.

GridSim [106] is a simulation toolkit developed in Java and based on the SimJava discrete event simulation package. It has been mainly used to model various scheduling and metascheduling approaches for grids. The infrastructure model consists of many grid sites with local schedulers which can be accessed using resource brokers. The toolkit has been applied to analyse various grid economy models, i.e. scenarios where resource providers charge their users for resource utilization. The toolkit takes into account network topology and energy-related costs. GridSim is available at http://www.cloudbus.org/gridsim/.

CloudSim [107] is a new simulation framework developed by the authors of GridSim. The resource model it implements is dedicated to cloud computing, where there are many cloud providers, each operating multiple data centres. The application model reflects the hierarchy introduced by virtualization: users can run virtual machines in data centres and execute application tasks on these virtual machines. CloudSim helps simulate various cloud computing scenarios, including public, private and hybrid IaaS infrastructures, as well as possible PaaS deployments on top of IaaS clouds. CloudSim is available at http://www.cloudbus.org/cloudsim/.

GSSIM is a grid scheduling simulator based on GridSim, which features several improvements. It provides a flexible framework for generation of workloads, resources and other events, and pluggable scheduler mechanisms for brokers and resource providers. It supports a job model based on the GRMS schema to model complex jobs; however job dependencies are not fully supported. GSSIM also provides repositories for workloads, plugins and experiments, in order to facilitate reusability and reproducibility of scheduling simulation experiments. GSSIM is available at http://www.gssim.org/.

GroudSim [108] is a recently developed grid and cloud simulator. In addition to performance improvements with respect to GridSim, GroudSim supports a hybrid grid and cloud infrastructure model. GroudSim is available at http://www.assembla.com/spaces/groudsim.

From among the toolkits described above, GridSim seems to be the most popular; however it requires many extensions and additional plugins which usually need to be developed for specific scheduling scenarios and algorithms. The recently-developed CloudSim and GroudSim packages implement a resource model which corresponds to cloud infrastructures. CloudSim is currently under active development and therefore appears to be the most promising from the cloud computing perspective.

4.2.7 Recommendations for VPH-Share

The specifics of the VPH-Share project are such that a number of workflows and applications are already available (or well into implementation) at the outset of the project, as described in [1]. It is important to note that most of these tools were not specifically designed for operation in the cloud, or indeed in any distributed environment – we are frequently dealing with standalone applications, using command-line interfaces and expecting to find their input data in local file systems. A key objective of WP2 is to enable developers to easily migrate and expose the functionality of their tools in the cloud. Thus, the deployment solutions need to meet two key criteria:

- migration must be associated with minimal development effort on the part of application contributors;

- a variety of cloud platforms, both public and private, need to be supported, given the diversity of market solutions and potential security constraints related to processing sensitive data.

This is why we have to discard wrapping and instrumentation solutions which are invasive (in terms of having to modify application source code), while preserving the generic nature of the deployment platform. A service wrapper seems to be preferable to compiling in WS interfaces for most of the tools and workflows contributed to VPH-Share.

We will therefore attempt to simplify the process of migrating existing application into the Cloud environment by using a generic framework which exposes command-line tools as web services so that they can be managed as Atomic Services and accessed by external tools. The SOAPLab framework seems to be the most promising technology given the dual requirements of genericity and robustness. Should command-line tool execution require additional logic (e.g. downloading input files from Cloud storage), a custom data access component would need to be preinstalled on the Virtual Machine on which a given application resides (this is further discussed in Section 4.4).

As far as deployment of applications on physical resources (supplied by cloud providers) is concerned, we need to provide an IaaS solution which would have the properties of a cloud federation. As described in Section 4.1.5 there are currently no “out of the box” cloud federation solutions; however by agreeing upon a selection of public and private cloud infrastructures we can engineer a tool which will be able to deploy Atomic Services on selected resources whenever requested by the workflow management/user interface components from Work Package 6. Since compatibility with EC2 has emerged as a de-facto standard in public cloud infrastructures, we will aim to retain it when acquiring resources from commercial cloud operators. The choice of middleware to support private cloud installations will be decided upon by Month 6 of the project, following recommendations from Task 2.2.

The resource demand estimation solutions presented are quite interesting; however none of them can be applied out-of-the-box. A black-box approach seems to be better suited for monitoring the load of servers hosting Atomic Services. However, the complex interaction model of Atomic Services prevents us from using log-based request-load correlation. Although instrumentation would provide more in-depth data on application performance, service instrumentation would need to occur for each Atomic Service, which is unfeasible – developers may wish to use arbitrary programming languages excluding the possibility of code reuse. Therefore, application-level instrumentation is not acceptable. VPH-Share should develop a mechanism which will reuse the best concepts from the presented solutions while fulfilling project-specific requirements. It is expected that Task 2.7 will provide information useful for understanding VPH-Share Atomic Service characteristics. For instance, the black-box approach could be applied to monitor only the load of virtual machines while instrumentation could be combined with the idea of wrapping legacy scientific applications, possibly implemented using various programming languages, with architecturally-aware interfaces [109], which, in the case of VPH-Share, would be HTTP-based. Woollard *et al.*

claim that the wrapping overhead is negligible. Such wrappers could be instrumented and reused for every Atomic Service, thus reducing developer effort.

Concerning, the final, but probably most innovative issue, i.e. deployment optimization heuristics, a survey of trends and solutions has been conducted. Among the presented approaches, the most promising ones seem to be Pareto-optimization combined with Constraint Satisfaction. Such a juxtaposition of optimization techniques would provide the means to optimize deployment using multiple criteria while preserving VPH-Share security constraints.

4.3 Access to high-performance computing environments

Several multi-scale computational workflows in the VPH-Share demand a large amount of raw processing power as well as a memory. In certain scenarios, the cloud computing platform is unable to provide this resource. Therefore a method is required to allow access to high performance peta-scale facilities such as Distributed European Infrastructure for Super Computing Applications (DEISA) [110] and Partnership for Advanced Computing in Europe (PRACE) [111] grid to provide these computational demands.

Traditionally, running an application on a single HPC often requires the user to ensure that the application is compiled and installed on the HPC, generate a job launching script for the queuing system and ensure that the data are correctly staged. This is often a straightforward, if not troublesome task. However, on a heterogeneous grid of super computers, the complexity grows significantly.

Grid computing [112] aims to simplify the access and usage of HPC resources which may span across multiple administrative domains and may be composed of many different types of resources. The software used to tie the grid together is often referred as the middleware. Currently, there are three popular heavyweight middleware which are Globus [113], Unicore [114] and g-lite [115].

A drawback of these heavyweight middleware is that they do not provide the simplicity and ease of use as envisioned [116]. Each grid may have its own security as well as deployment constraints and procedures. The high barrier of the technological expertise to install and deploy applications across grids has discouraged many scientists from adopting this technology is described by Chin *et al.* [117].

A method of providing simplified access to HPC resource is required. Furthermore, the solution also requires the integration with the VPH-Share cloud framework including the components developed in Task 2.1 and Task 2.2, access to data storage developed in Task 2.4, the security framework developed in Task 2.6 and VPH-Share application developed in Tasks 5.4 to 5.7. The master interface design as well as workflow execution in Task 6.4 and 6.5 respectively will also have to be considered in Task 2.3.

4.3.1 State of the art assessment

The complexities of deploying software on grids have led to the development of a lightweight middleware solutions such as the Application Hosting Environment (AHE) [118], GROWL [119], JavaGAT [120] and Styx [121]. The aims of these frameworks and APIs are to simplify and hide the complexity of the traditional heavy grid middleware.

AHE is based around the idea of grid application virtualization where scientific applications can be easily exposed as web services, compliant with the WSRF specification [122]. This allows applications to be deployed on a number of different types of resources from high performance grid machines to simple desktops. AHE is a lightweight middleware that exists on top of existing grid frameworks such as Globus and Unicore. It abstracts the complexity of the underlying infrastructure and provides services which allow researchers to manage their workflow including the running of their simulation, file staging and retrieval as well as security management. In the area of application deployment, AHE promotes the community model where "expert" scientist would configure the AHE to deploy their scientific applications and use the AHE to share their application with their community. Once community member receives the AHE client, they can simply install the light weight client and launch and monitor the application across the grid.

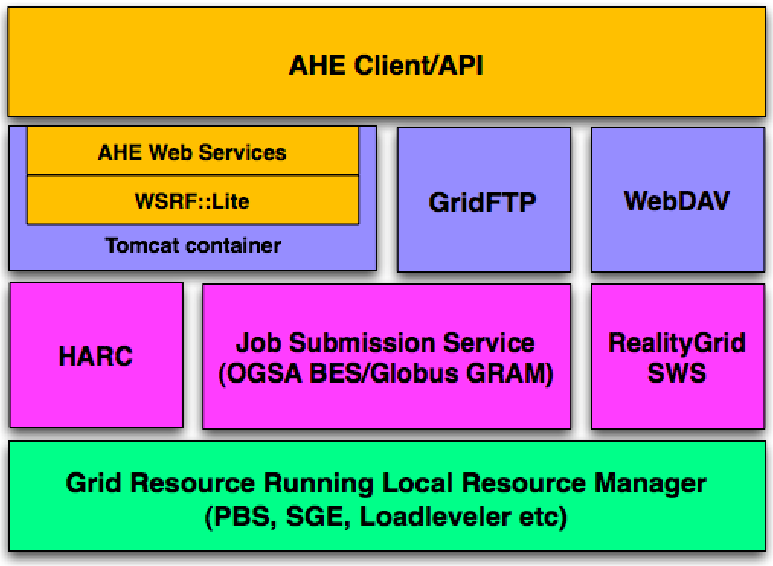
AHE Client/API
AHE Web Services
WSRF::Lite
Tomcat container
GridFTP
WebDAV
HARC
Job Submission Service (OGSA BES/Globus GRAM)
RealityGrid SWS
Grid Resource Running Local Resource Manager (PBS, SGE, Loadleveler etc)

Figure 7: The AHE Architect is composed of many components. Including client/API, HARC, RealityGrid Steering System, resource manager and job submission and web services

AHE is designed based on the Service Oriented Architecture. The overall architect can be seen in Figure 7. It is built upon WSRF::Lite [123], a Perl implementation of the OASIS web service resource framework built upon the Perl SOAP::Lite web service toolkit [124]. GridSAM [125] is used as the web service interface which is used to accept job submission as well as monitor the jobs. GridSAM also allows applications to be launched from a number of different distributed resource managers such as Globus or Sun Grid Engine [126]. AHE provides features such as computational steering where simulations can be monitored and modified in real time. This is implemented using the RealityGrid steering system [127], which is a WSRF-compliant middleware library that allows parameters in simulations code to be marked steerable. This can be particularly helpful allowing researchers to view and adjust parameters and prevent valuable computational time to be wasted. Another feature of the AHE is the advanced co-reservation of resource which allows a uniform access to reserve time across different resources; this is achieved through the Highly Available Robust Co-scheduler (HARC) [128]. HARC works by deploying resource manager components on to the computational resources allowing the user to query and issue commands to that resource. HARC utilizes the Paxos Commit protocol [129], the main advantage of this is to remove the problem of transaction manager failure experienced by two phased commit protocol. This allows the system to function as long as the majority of transaction managers are functioning. The main advantage of HARC is to provide interactivity to the user as well as a schedule "first class" grid resource [130].

AHE provides a standard compliant submission system using the Open Grid Services Architecture (OGSA). This includes the Basic Execution Service (BES) which employs the Job Submission description Language [131] allowing it to work with Unicore version6 and GridSAM. AHE was also developed to submit jobs to Globus 4 GRAM. This flexibility allows AHE to access a number of different types of computing grids through a single user interface.

The current AHE security framework is built around the Audited Credential Delegation (ACD) [132]. ACD was developed in mind to secure VPH related projects by providing authentication, authorization and auditing in distributed VPH project environment that would remove digital certificates from the security mechanism from the end-user point of view. The lengthy process of obtaining digital certificate still has to be completed but by an expert user only once, this considerably simplifies the usability of the AHE [133]. Many current security implementations use Public Key Infrastructure (PKI) and X.509 digital certificates [134] to provide authentication and authorization. An existing problem of current security mechanisms is the complexity and time consuming task for both the administrator and the end users [135] [136]. An example of such a problem includes the lengthy process of acquiring a X.509 certificate and the generation of proxy certificate to access remote resource. This task often led to users sharing private keys which weaken the security of the environment.

ACD is designed around the concept of "wrappers", these wrappers delegates the transactions between the virtual organization (VO) and the outside world. Any actions or requests are intercepted by the security wrapper where the identity of the user is established, the authorization of the action or request checked, the result of these security checks logged and finally returned the results to the user. ACD is designed round web services using standards such as Web Service Description Language (WSDL), SAP, WS-Policy and WS-Security [137]. ACD has four main components: a local authentication service (LAS) which removes the need of digital certificate from the end-users' experience using username-password combination. An authorization component that controls all action performed within the virtual organization. This is achieved by using the Parameterized Role Based Access Control (PRBAC) model in which permissions are assigned to roles [138]. The authorization component only manages the permissions given by the resource provider which has the final say on what action is allowed within their infrastructure. The third component of the ACD is the credential repository, this stores the certificates required to communicate with the grid. An expert user of the VO is required to obtain the certificate and set it up only once. To allow members of the VO to access the grid, ACD issues digital certificate which authenticates requests by the VO to the resource provider. This component also maintains a list of proxy certificates, the corresponding private keys and the association between user and proxies. This mechanism allows ACD to record the user action. The last component is the auditing component which records all action within the VO by the users. A typical security handling of a user action can be seen in Figure 8.

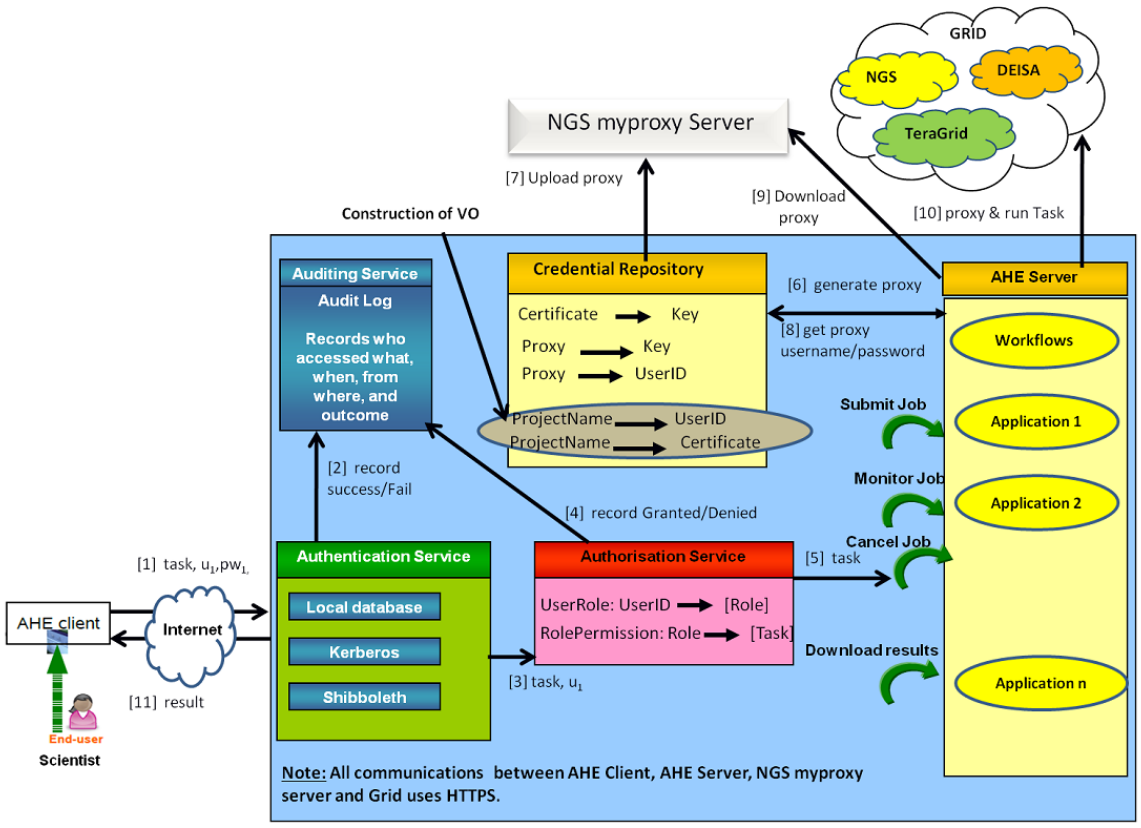

Figure 8: Typical steps involved in handling user actions by the ACD security mechanism in AHE

The AHE-ACD framework offers the following administrative actions including: Create VO, Assign Certificate to VO, Add user to VO, reset user password, create role, assign tasks to role, assign users to role and functional actions including: *prepare job*, *submit job*, *monitor job*, *download* and *terminate job*.

Grid Resources on Workstation Library (GROWL) is a lightweight and extensible toolkit which provides an interface to existing grid surface for applications developed in R, C and FORTRAN. GROWL employs the gSOAP web service which is further wrapped by GROWL API. To deploy the GROWL framework, a user selects the component that he needs and builds the GROWL library. Once compiled, any calls to the GROWL library will be automatically redirected to the GROWL server which then redirects the job to the appropriate resource. GROWL provides an authentication service by using proxy with their GSI certificate deployed into a MyProxy [139] server. The GROWL server can then request a delegate certificate allowing the server to act on-behalf of the server.

GROWL's main focus is to provide a lightweight and extensible framework. It however, lacks many features such as steerability, resource management as well as robust job scheduling methods.

The Styx Grid Service [121] is a lightweight middleware for scientific workflow by wrapping command line programs and deployed as web service. This allows them to run across the internet as if they were local programs. The core of the Styx Grid Service is the Styx protocol for distributed system. This protocol is a file-sharing protocol which treats all resources as files. These files are organized into a hierarchy known as namespaces. This allows all resources using Styx to be represented as URL allowing data to be directly shared between different web services. For security, Styx uses TLS as well as public certificate authentication. The Styx Grid Service can be exposed as a "WS-Resource" allowing it to maintain states across different web-service invocation.

The Styx Grid Service has been tested with the Taverna workbench workflow system [140] as well as the Triana workflow system [141] with some modifications. It does, however, lack features to control and interact with common grid middleware.

The Java Grid Application Toolkit (JavaGAT) is a high-level and middleware independent API for the grid. As the name suggests, JavaGAT is a Java implementation of the GAT [142] specification. JavaGAT is a feature-rich API with the aim of simplifying access to multiple domains of grids for developers. It supports resource management, security, grid I/O and application steering and monitoring of different grids through JavaGAT adaptors as well as novel features such as nested exceptions and intelligent dispatching of method invocations to handle errors and selecting suitable grids for requested operations.

JavaGAT is designed with portability in mind, by using Java for development and deployment of the grid application. It is capable of selecting the best grid for deployment – a feature known as "intelligent dispatching". JavaGAT deals with fault tolerance through nested exception which detects unavailability and failures of an implementation, although it does not report failures that are not detected by the middleware itself. A nested exception is thrown when all JavaGAT adaptors fail, informing the user of the error details.

Some notable features of JavaGAT API include grid I/O support for many different types of grid middleware, although complete support for I/O for different middleware platforms remains incomplete. The JavaGAT API provides resource management so as to control how resources and binaries are pre-staged and post-staged. Different grid middleware suites handle this differently and in certain situations results may be lost or overwritten when multiple jobs are submitted to the same middleware. To overcome this, JavaGAT introduces a sandbox mechanism on the remote machine where all application files and directories are copied before the job is started, and where the results can be stored prior to retrieval.

JavaGAT provides several security mechanisms including caching passwords and credentials as well as supporting MyProxy credential management service.

Other similar grid APIs include Java CoG [143], Simple API for Grid Application (SAGA) [144], DRMAA [145] and the GridRPC specification [146]. These APIs provides low-level simplified access to different grid middleware platforms for developers. However, due to the requirement of WP2, these APIs are not well suited for workflow tools and will not be discussed in detail.

4.3.2 Recommendations for VPH-Share

A number of technologies have been presented, including AHE, Styx and GROWL which target the deployment of applications across a grid, as well as JavaGAT, a Grid API which simplifies the development of applications for the grid.

In the context of the VPH-Share project, a lightweight middleware solution is required for computational workflows to access HPC resources which cannot be run on the cloud computing platform. This calls for a solution which can be easily adopted or with minor modification to existing simulation tools as well as extensibility to support seamless integration with the resource manager in the cloud platform as well as the security platform which will be deployed in WP2.

The Application Hosting Environment has been previously used in other projects such as Virolab [147] and other simulation codes such as NAMD [148], CHARMM [149], LAMMPS [150], VASP [151], DL_PLOY [152], LB3D [153] and HemeLB [154]. The AHE application virtualization paradigm allows complex, multi-component grid based application to be represented as a web services. It also contains a number of features such as steering and advance co-reservation, along with a user-friendly security framework. Some of these features are not present in GROWL or Styx Service Grid.

Another potential candidate is the JavaGAT API. This API contains many features such as error message logging, intelligent dispatching and support for different grid middleware suites. However, this API is targeted at developers and requires existing applications to be re-implemented using the respective APIs. This would require significant development work which is unsuitable for the scope of this task.

There are several areas of development associated with AHE in this task. One area is integrating AHE with the cloud platform’s resource manager as well as providing seamless interaction for modelling tools between the cloud and HPC platforms as part of the components developed in Task 2.1 and Task 2.2. AHE can also be deployed as a standalone platform providing flexibility of workflow-user interaction, designed as part of Tasks 6.4 and 6.5. To facilitate this development, the AHE API may have to be refactored to be compatible with the VPH-Share cloud-based Atomic Services and the underlying resources. Furthermore, to integrate AHE into the VPH-Share cloud platform it will have to report load information for HPC resources to the resource managers developed in Tasks 2.1 and 2.2. AHE features will also need to be extended to cover data staging/access from facilities developed in Task

2.4 for data storage. The ACD security framework in AHE will have to be integrated into the overall cloud security framework developed in Task 2.7.

Another area of development is to ensure compatibility between VPH-Share applications and AHE. To achieve optimal efficiency, these tools may also have to be optimized for specific grid middleware. Requirement gathering and analysis will have to be performed on modelling and simulation tools which will require HPC, identified in Tasks 5.4-5.7.

There are also a number of areas in which AHE can be improved, including usability and error tracking and handling. Requirement gathering from workflow tools as well as coordination with modellers, UI designers and cloud architect is required to ensure that AHE delivers the required functionality.

4.4 Access to large binary data in the cloud

Just like cloud computing, cloud storage has also been increasing in popularity for many of the same reasons as cloud computing. Cloud storage delivers virtualized storage on demand, over a network based on a request for a given quality of service (QoS). There is no need to purchase storage or in some cases even provision it before storing data. However, in the scope of VPH-Share it cannot be assumed that data will be located in a single cloud storage infrastructure. Moreover, for security reasons, institutes and hospitals may opt for private cloud storage, or prefer to use their existing storage infrastructure. In the case of data-intensive applications and HPC, although computational and storage resources may be abundant, access to large data sets often proves a significant drawback. Thus, to transport, access and process large data in the cloud, efficient protocols and mechanisms must be put in place.

4.4.1 Cloud storage concept and services

Almost all cloud storage services, use the following concepts: *container*, *folder*, *data object* and *virtual path* to present and allow users to manage their data. Figure 9 shows the relation between these concepts.

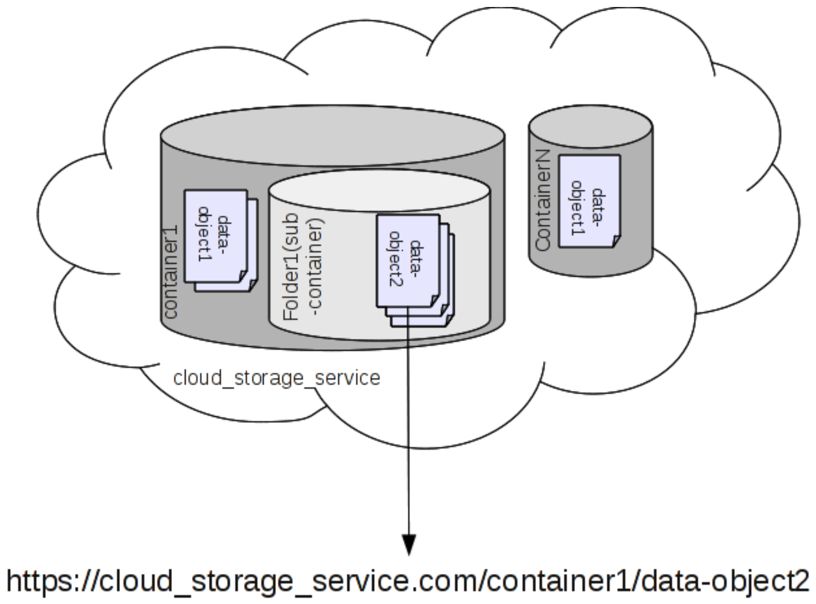

Figure 9: Relations between container, folder and data object

A container is a namespace for folders, data objects and virtual paths. Depending on the cloud storage service, the scope of the namespace can be global, account, or sub-account. For example, Amazon S3 containers are called buckets, and they must be uniquely named so that the namespace is global. In other cloud storage services, the naming convention of the container is less strict. All cloud storage services allow users to list containers and contents within them. These contents can be either data objects, and depending on the storage service, folders, or virtual paths.

A data object consists of unstructured data residing in a container. Some cloud storage services call them objects, blobs or files. Users can look up data objects in a container with the use of keys, which often relate directly to the URLs used to manipulate such objects. A data object can be of zero or greater length. Some services restrict the size of a data object to 5GB. Finally, data objects can have metadata in the form of key-value pairs that can be stored alongside the data. When a data object is inside a folder, its name is relative to that folder.

A folder is a sub-container. It can contain data objects or other folders, depending on the service. The names of items in a folder are base-names. Data object names incorporate folders with the use of separators ("/") – in the same way as in a regular file system. Virtual paths, on the other hand, are purely used to create the appearance of a hierarchical structure in flat cloud storage.

Everything in a cloud storage service is stored in a container. As mentioned earlier, a container is a namespace. Thus, if a container's name is *container1*, it will be accessed via a URL. For example, if a cloud storage service is called *cloud_storage_service.com*, a container in that service would appear as: https://cloud_storage_service.com/container1. If the name of a data object is stored with the key *data-object2*, the URL pointing to that data object would be: https://cloud_storage_service.com/container1/data-object2. Although naming conventions may change from one service to the next, the general approach is the same.

By default, every item in a container is private. If the user wishes to grant access to others, he has to explicitly configure it. Currently, the means to expose containers to the public are provider-specific. Finally, cloud storage services, provide versioning for data objects.

The Amazon Simple Storage Service (Amazon S3) is Amazon's cloud storage service. It is built with a minimal set of features so that the focus is on simplicity and robustness. Amazon S3 stores data as objects within buckets. A maximum of 100 buckets per account can be created and stored. An object is comprised of a file together with its metadata. The maximum size of an S3 object is 5GB. To store an object in Amazon S3, the user uploads the file he/she wants to store to a bucket. When the file is uploaded, the user can set permissions on the object as well as any metadata. Additionally, Amazon S3 provides access control over buckets, and enables users to view access logs for buckets and their objects. Moreover, the users can choose the geographical region where Amazon S3 will store the bucket and its contents. In Amazon S3, by default the bucket owner pays for downloads from the bucket. Alternatively, the owner can configure bucket parameters, so that third persons requesting to download contents from that bucket are charged. Since in Amazon S3 downloads are charged for, Amazon provides the ability to obtain objects via BitTorrent and save bandwidth on large files. Moreover, for large files (larger than 100 MB) Amazon S3 provides multipart upload capabilities, allowing users to upload a single object as a set of parts. Each part is a contiguous portion of the object's data.

Another popular cloud storage service is RackSpace Cloud Files. The way RackSpace Cloud Files organize storage is no different from Amazon S3. In this cloud storage service, the topmost concept is the container, which is used to hold objects. However, containers in RackSpace cannot be nested. Similarly to Amazon, RackSpace imposes a limit on the size of a single uploaded object, which is 5 GB. Moreover, RackSpace has no permissions or access control for containers or objects. Each user owns a private storage account and has full access to that account. The only way a user can access the content from another account is if he shares credentials or a session token. For downloading and uploading data, RackSpace uses HTTP.

4.4.2 Federated cloud storage

Federated cloud storage refers to the aggregation of multiple cloud storage resources. Figure 10 shows an aggregation of cloud storage resources. In this case, the aggregation will be realized from the client's point of view. This means that a common management layer will govern loosely coupled storage resources. Through this common management layer, a multitude of private and public clouds can be aggregated exposing all storage as a single name space. This unified view will provide benefits, such as:

- shared storage between cloud storage offerings;
- distributing data across the cloud to ensure accessibility and reduce latency;
- dynamic addition and removal of storage providers;
- providing virtually limitless storage capacity;

- improving collaborations through data sharing.

Federating storage however, comes with some challenges. To implement a common management layer for a variety of storage systems requires a high-level programmable interface. This interface will hide the complexity and specifics of accessing different storage systems. Many cloud providers have licensed their proprietary APIs freely, allowing anyone to implement applications that make use of their infrastructure. As already mentioned in the previous sections, despite the accessibility of such APIs, their use is problematic. The complexity of developing against many different APIs is a key problem. Additionally, the usage of proprietary APIs will bound development cycles to the standards imposed by vendors. Finally, the flexibility of adding and using new storage resources and technologies will be limited.

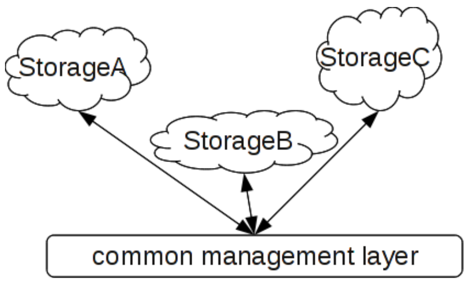

Figure 10: Data cloud storage

4.4.3 Transport protocols

Many cloud storage services have emerged in recent years. However, moving large amounts of data to these facilities remains a bottleneck. The concern about public clouds is transfer costs. For private and public clouds, network capacity and the choice of transport protocols is the limiting factor. Uploading 100TB+ data to an online storage is still less than practical in most cases. Therefore, while storage capacity in the cloud is expandable, limits in the capacity of network connections to the cloud can create challenges for scientific application with multiple petabytes of data to be moved back and forth. In most cases data transfers between clients from cloud storage, are facilitated through HTTP(s). Although HTTP(s) is a well-established and reliable protocol, other solutions may be more suitable for bulk data transfers. Moreover, direct client to cloud data transfers, may be suboptimal due to the network latencies on the client side.

4.4.4 State of the art assessment

To this date, most of the proposed solutions for cloud federation, concern the supply side of public clouds. Numerous architectures have being proposed on how providers can merge their infrastructure to meet spike demands. Moreover, these solutions are concerned with scalability problems from the provider's point of view [155] [156] [157]. Additionally, these proposals mostly concern computational clouds. Far fewer proposals try to reduce the client-side cost of using federated clouds [158] or offer clients a unified view of their cloud storage [159]. However, some proposals have being made in order to improve cloud storage on a more interoperable client-centric way [160] [161] in order to improve fault tolerance on the cloud.

Not many publications address the issue of data transport protocols for the cloud. However, major cloud players like Amazon have looked into the performance problems of transporting data. An explanation of the data transport problems involved in cloud storage can be found by Graham-Rowe [162]. TCP seems to be the bottleneck for bulk data transports. While ideal for moving small amounts of data through the Internet, TCP is less suitable for larger datasets, with large round-trip time (RTT). Amazon addresses this problem by using a new protocol called the Fast and Secure Protocol (FASP). FASP was developed to accelerate bulk data movement in the face of large RTT and severe packet loss with the use of UDP. FASP, however, remains a proprietary protocol. UDT [163] is another protocol interesting from the point of view of rapid data transfers to and from the cloud. UDT has been developed by Sector storage cloud [164] to provide fast data transfers. Besides high-performing protocols, optimized data management and placement can increase data transport performance. Authors of [165] propose a service-oriented resource broker (SRB) to ensure QoS and optimized bulk data transfers.

4.4.5 Recommendations for VPH-Share

The main problem in using different cloud storage systems is the lack of standards. What is needed is a vendor-neutral, standard API for cloud computing that all vendors can implement. This will allow developers to easily integrate storage from different cloud vendors. This approach will avoid lock-in and reduce storage costs. There are several efforts already underway to do just that, as described on the Cloud Standards website [166]. The Open Cloud Computing Interface (OCCI) [58] seems more suitable for cloud data storage as several API implementations already exist. OCCI specifications are delivered through the Open Grid Forum (OGF) [167]. OCCI is a set of specifications for cloud-based interactions with resources in a way that is vendor-independent and platform-neutral. OCCI provides a flexible API with strong focus on integration, portability, interoperability and innovation while still offering a high degree of extensibility. Jclouds [168] is an OCCI implementation supporting many clouds including Amazon, GoGrid, Azure, vCloud, and RackSpace. The most interesting part of this API is the BlobStore API, providing the means for managing key-value storage providers such as Microsoft Azure Blob Service and Amazon S3. It offers both asynchronous and synchronous APIs, as well as map-based access to data. Occi4java [169] is another implementation of the OCCI specification – an open-source, Java-based implementation of OCCI on top of the libvirt Hypervisor Abstraction API. This API offers a Maven-based loose coupling of all three specification documents and supports all possible actions on computing resources and several actions on network and storage resources.

Moreover, if multiple cloud storage resources are to be made available, a file-system view would be very useful. To enable this, a virtual file system implementation will transparently integrate multiple autonomous storage resources providing virtually limitless capacity. As a result, a file system as a service can be provided, optimizing data placement, storage utilization and accessibility.

In the scope of transport protocols, the key obstacle to rapid data transfers in the cloud seems to be TCP. As stated in Section 4.4.3, effort is underway to provide fast data transfer through high-performance protocols. Together with these transport protocols, service-oriented resource brokers can optimize data transfer rates. To avoid centralization bottlenecks, connection services can be deployed next to or near the data.

4.5 Data reliability and integrity

Data reliability and integrity is needed to ensure sound use of the biomedical datasets manipulated with the use of VPH-Share applications and tools. Simulations results and inferred medical outcomes must be based on reliable data. Today's cloud delivery models do not offer means for the cloud user to perform such auditing tasks in a certified and trustworthy manner. An interesting discussion of the current state of the art with respect to cloud data management can be found by Jeffery and Neidecker-Lutz [2]. As noted there, there are extant concerns over availability and business continuity in the context of cloud usage – with some recent examples of failures (see for instance the failure of Swissdisk [170]); moreover, there are few studies regarding the actual usage patterns for file and data access in cloud systems [171].

As the biomedical files may be very large and need to persist for long periods, special reliability and integrity mechanisms should be enforced on top of cloud storage. The WP2 infrastructure needs to be able to perform the following tasks:

- periodic integrity checks on data objects with the use of hash algorithms;
- facilitating storage of multiple copies of data on various cloud platforms;
- tracking the history and origin of binary datasets.

In order to meet the above goals, the storage cloud object model and interfaces must be extended, supporting multiple objects copies over multiple clouds and execution of periodic processes over the storage cloud infrastructure. We intend to approach this issue by introducing the concept of a *managed dataset*, which represents a specific data item which is known to the VPH-Share services and for which specific information can be located in the VPH-Share metadata registry. Managed datasets can be registered with VPH-Share upon being uploaded to the cloud infrastructure – subsequently the infrastructure should be able to monitor and track their availability in an automatic manner. Thus, the functionality provided by Task 2.5 is intimately tied to the data access layer provided by Task 2.4.

In order to assess the state of the art in the area of ensuring the integrity and availability of binary data stored in a cloud framework, we need to refer to the storage mechanisms supported and exposed by major cloud computing platforms. A number of solutions have been developed to deal with the problems of data integrity, availability and retrievability in cloud computing environments; however prior to discussing specific solutions to such problems, it is necessary to define each term in more detail. Thus:

- Data integrity is the measure of whether the data stored in the cloud remains in an unaltered state and is visible as such to whoever retrieves it. Data integrity is not directly tied to availability concerns; rather, it deals with potential data corruption (intentional or otherwise) that may affect objects stored in the cloud.

- Data availability is the measure of whether a user who has a legitimate claim on a given dataset can actually access the data in question. Technical failures may contribute to reduced availability of data objects in distributed infrastructures such as the cloud. Data availability is typically achieved through redundant resource utilization which enables the infrastructure (as a whole) to mask technical failures without affecting the end user.
- Data retrievability deals with the practicality of retrieving data sets for use in computations. Certainly, a large data set – even if completely intact and readily available – is useless if it cannot be efficiently fed into a computation (or, conversely, if the computation cannot be migrated to the host on which the input data resides). This issue is of particular importance in distributed environments where data and computations need not be geographically proximate. Data retrievability is intimately tied to the issue of load balancing, addressed in Task 2.2 of the VPH-Share project.

4.5.1 Data storage reliability and availability in commercial clouds

While commercial cloud providers usually boast high availability and reliability of data storage resources, they typically provide little end-user control over the low-level mechanisms used to enforce such properties. Internally, providers apply a range of mechanisms (mostly based on replication and resource redundancy) to ensure smooth and timely access to data; however due to the lack of industry standards regarding cloud data access, particularly with respect to binary data, there is currently no system which would enable cross-cloud binary data management.

The Amazon S3 storage system comes with availability guarantees which profess 99.99% availability and carry a service level agreement providing service credits if a customer's availability falls below 99.9% [172]. Data is distributed across storage resources assigned to a specific geographical region. Of note is the fact that data integrity guarantees vary from region to region (for instance, read-after-write consistency in the EU region and eventual consistency in the US standard region). There are no specific mechanisms for assessing the retrievability of binary data, although the vendor promises inbuilt protection against transfer spikes.

Internally, the Amazon data storage services (including S3) are based upon a proprietary key-value storage system called Dynamo [173]. Dynamo uses a synthesis of techniques to achieve scalability and availability: data is partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo employs a gossip-based distributed failure detection and membership protocol. Storage nodes can be added and removed without requiring any manual partitioning or redistribution; however all this functionality is concealed from the end user and cannot be directly influenced by applications making use of the Amazon data storage services.

RackSpace, another popular IaaS, provides a solution called Dedicated Network Attached Storage [174] with inbuilt disaster recovery and data replication mechanisms. The provider is capable of restoring missing data on request (this mechanism is called Unmetered Managed

Backup); however, no automatic recovery mechanisms were in place at the time this deliverable was published. Data retrievability is not covered by the RackSpace SLA.

GoGrid provides a Cloud Storage solution [175] which enables automated management of binary data. Data sources can be mounted via a secure private network on Linux and Windows machines and use common transfer protocols to move data to and from the Cloud Storage device. The infrastructure provides usage reports but does not make promises regarding the consistency and availability of data stored in the CS framework. Backup is file-based and depends on decisions made by the end user.

The Microsoft Azure [47] platform facilitates storage of large binary objects (BLOBs) and replicates them internally on three separate fault domains, thus ensuring data availability. Operations on binary data are executed via a RESTful interface; however the application cannot directly influence the placement and retrievability of binary objects.

4.5.2 Improving cloud data availability and integrity – on-going research initiatives

As the leading commercial cloud platforms typically do not provide explicit mechanisms for managing the availability and retrievability of binary data, a number of add-on solutions and concepts have recently been developed. One of the interesting approaches to solving such issues comes from the cryptographic community, where the concepts known from public key infrastructures (PKI) can be reused to provide practical and theoretically verifiable methods of proving the required properties of data storage, together with appropriate protocols for message exchange between storage providers and its clients.

Juels and Kaliski [176] propose the concept of proofs of retrievability (PORs) for large files. A POR scheme enables a storage provider (prover) to deliver a proof that a user (verifier) can retrieve a file and recover it in its entirety. The POR scheme is specifically designed to handle large files, such as backups, which do not necessarily have to be frequently read; however, their integrity and availability has to be assured. The proposed POR scheme requires the administrator to insert into the file, and subsequently distribute some additional, randomly-generated data, called sentinels, prior to storing the file in the archive. Information about sentinels is securely stored and periodic checks can be performed, which, with high probability, reveal whether a storage provider has deleted or modified a portion of the file.

Bowers *et al.* [177] describe a solution called HAIL (high-availability and integrity layer) for cloud storage, which is based on the POR concept. The HAIL concept is similar to the RAID solutions known from storage servers. The authors extend it with a protocol using such techniques as message authentication codes (MAC) and error correction codes (ECC) for providing reliable replication of data with PORs. File data and parity code is distributed across multiple servers for redundancy, which protects the file against mobile adversaries who can corrupt all servers, but only *b* out of *n* servers at a given time.

A different approach is pursued by the VENUS system (VErificatioN for Untrusted Storage) [178]. In VENUS, cloud storage is treated as a single trust domain while clients who collaboratively access data storage provide integrity and consistency.

4.5.3 Recommendations for VPH-Share

Owing to the large variety of data storage protocol and access methods, ensuring integrity, availability and retrievability of medical data requires a dedicated mechanism provided by the Project itself. From the state of the art it seems that we can make no assumptions regarding the level of protection and redundancy of binary data deployed to the cloud and that the only mechanisms which are universally available to end user involve actual storage and retrieval of data items. As VPH-Share aims to provide a cross-cloud computing environment where computations can be delegated to a variety of computing infrastructures, both public and private, it is important to deploy a solution which would provide consistent data availability guarantees regardless of the underlying storage system. Hence, Task 2.5 defines the concept of a *managed dataset* (as presented above) and will implement mechanisms to ensure continued access to such datasets, once they are registered with the metadata registry provided by Work Package 4. We foresee an automated component, itself part of the WP2 platform, which would periodically check for the availability of registered datasets as well as for the integrity of the actual data (through a checksum-like mechanism). Through the use of data manipulation tools developed in Task 2.4, the component would make autonomous decisions regarding migration and replication of data, for instance if access time proves insufficient or if the user tags a given dataset as particularly crucial. Moreover, the WP2 platform – by interposing itself between the actual application and the requested binary data – will be able to keep track of the frequency with which individual data elements are accessed, as well as the credentials of users requesting (and potentially obtaining) access to such data. This information would enable tracking the history and the origin of the data and who had custody of it over time for auditing purposes, supporting multiple objects copies over multiple clouds and execution of periodic processes over the storage cloud infrastructure, as defined in the Project's Description of Work [1].

4.6 Security for cloud applications

In this section we describe state of the art solutions for securing distributed computing systems and the secure management of virtualized resources.

4.6.1 Definitions of security terms in a virtualized environment

Before starting with technologies, designs and challenges of Security for Virtualized Environments, we should first establish how we define the main security terms in this context.

According to European Commission's "Trust and Security" Unit final report on Cloud Security, Security includes the confidentiality, availability and integrity of data or information, as well as authentication and non-repudiation. Privacy concerns compliance with various regulations regarding the right to private life (in our context, European data protection regulations). The globally accepted privacy principles we should take into account include: consent of use, purpose restriction, legitimacy, transparency, data security and data subject participation. Trust is the "assurance" that people, data, entities, information or processes will function or behave in expected ways. Trust might be regarded as a consequence of progress towards security or privacy objectives.

In a virtualized environment, while these definitions are completely valid, addressing the risks might be a difficult task because of some challenges, specifically in policy management and enforcement. While specific security policies for applications or services might be verified successfully in a "traditional" environment, the policy combination in a cloud environment might degrade their strength. Control of the consent of use and privacy (control of personal data) in an implementation of cloud computing is certainly more complex, as the users' and even the cloud providers' control of personal data must be establish across an infrastructure that opaquely and autonomously distributes it.

Currently, encryption of the data is the best way of maintaining confidentiality, but still that doesn't provide the user with means of preventing its distribution and use under policies which are different than the ones the user "expected" its data to be accessed.

This brings us to compliance verification with regulations. Cloud users may have to take on trust their providers' compliance with the required regulations and standards, with none (or almost none) of the compliance verification tools and mechanisms that are available in more traditional environments.

Given the complexity associated with cloud computing and its distinct challenges, security monitoring for distributed computation environments is key to secure data, provide privacy and prove the compliance of the implementation with regulations to the users.

4.6.2 Network and OS security

Although this subject is not unique to clouds, it is an essential part of ensuring security on this level for cloud applications running on IaaS clouds. Those types of clouds are the most important for the VPH project due to its requirements. Additionally, some aspects related to network security in public clouds differ from those in which we have full control over our infrastructure and, as such, should be enumerated here.

4.6.2.1 Network security

By default, (unless special services are offered, as shown later) in the IaaS model cloud providers are responsible only for mitigating (if possible) basic threats against their network. This especially includes attacks on network availability such as (Distributed) Denial of Service (D)DoS. However, they do not ensure confidentiality or integrity of the transmitted data, as such features are normally not provided by lower layers of the OSI model. Typically, those are covered by upper-layer protocols such as SSH or TLS. However, occasionally, applications which cannot be replaced may use insecure (or insufficiently secure) protocols. To allow usage of those applications while maintaining an adequate level of security, it is necessary to provide appropriate mechanisms on lower layers.

There are numerous solutions fulfilling such requirements. An examples of the most straightforward solution is IPSec [179]. It has been created for the IPv6 protocol and subsequently backported as an extension for IPv4. As such, it uses raw IP packets with additional headers – Authentication Header (AH) [180] and Encapsulating Security Payload (ESP) [181]. This enables IPsec to provide integrity, authentication and confidentiality of the payload (and some header fields), as well as prevent replaying packages (resending captured packages). For proper operation, the protocol requires secure exchange of a shared secret. This can be done manually, but in most cases would be impractical. Hence, the specification references automatic key exchange mechanisms. By default, Internet Key Exchange Protocol Version 2 (IKEv2) [182] is used. IPsec, much like IP, may encapsulate multiple protocols such as transport protocols (like TCP or UDP) and other IP protocols. This allows IPsec to operate in two modes – so-called transport mode, in which IPsec directly secures upper-layer protocols in the network, and the tunnelling mode, which allows creation of tunnels (VPNs) e.g. between border routers of two LANs through the Internet, by encapsulating IP in IPsec.

IPsec has many merits – it provides a good level of security using native IP packages without re-encapsulating significant portions of the TCP/IP stack, which lowers overhead. Additionally, it is based on the IP protocol and thus may be used across the Internet without significant problems (in most cases). However, there are also drawbacks, the most important of which relates to the fact that some network firewalls may be configured to filter out (for security reasons) all non-standard IP traffic (e.g. traffic encapsulating protocols other than TCP, UDP and ICMP). Such configurations would then block IPsec as AH/ESP packages would be dropped as well. There are two solutions to mitigate this problem:

- IPsec packages cloud be encapsulated into different ones, such as Layer 2 Tunneling Protocol (L2TP) [183] running on top of UDP;
- Cloud providers may offer a dedicated IPsec solution, which, of course, would require them to appropriately configure their network (so it could accept IPsec packages). A notable example of such a service is Amazon Virtual Private Cloud (VPC) [55], which, apart from other features, enables creation of pure IPsec tunnels between the client's network and Amazon;

Other solutions exist, such as:

- OpenVPN [184] allowing encapsulation of IP packages or Ethernet frames in TCP or UDP packages. It provides secure transfer of the data by means of authentication and encryption of encapsulated payload. OpenVPN may use pre-shared secret or PKI infrastructure;
- Encapsulation of Point-to-Point Protocol (PPP) into SSH (application layer) protocol – payload is protected by the security mechanisms of SSH;
- IP over HTTPS (IP-HTTPS) Tunneling Protocol [185] – protocol proposed by Microsoft; enables tunnelling IPv6 (but not IPv4) packages over HTTS – its goals are to allow IP communications in extreme network conditions where all other protocols that may be used for tunnelling are not available. Security of communication is ensured by TLS.

All of the mentioned protocols operate on high enough OSI layers that there should be no problem with communication with cloud instances on the cloud provider side. If the client's network permits, OpenVPN would be the best choice for VPH-Share as it offers the lowest overhead.

4.6.2.2 OS security

In IaaS, the cloud user is fully responsible for maintaining the OS running in the cloud. This process generally isn't much different from non-cloud models and includes:

- Ensuring that the installed software is up to date;
- Use log analysis scripts such as Fail2ban [186] or full (host-based) Intrusion Prevention and Detection Systems (IDS/IPS) such as Snort [187];
- Checking for software vulnerabilities, possibly with help from vulnerability scanners such as Nessus [188] or its Open Source variant – OpenVAS [189].

In clouds, the scale of operation is usually much higher, but at the same time atomic service instances are typically based on more or less homogeneous templates, and do not run for long. Hence, it is typically enough to update software in templates rather than on operating nodes.

4.6.3 Related cloud technologies and architectures

In this section we discussthe security provided by the four cloud stacks presented in Section 4.1.2: Eucalyptus, OpenNebula, OpenStack and Nimbus, any of which could be used for private cloud application deployment in VPH-Share. Additionally we've analysed security features of selected public cloud providers.

We also provide an overview of the architecture proposed by the OpenTC consortium, as especially relevant for the VPH-Share Work Package 2. The sHype framework referenced here proposes policy management that goes beyond "mere" access control. Such a policy management would effectively provide us the means to secure deployed VMs and distribute requirements and constraints throughout the system. Finally, Trusted Virtual Domains gives us a model to secure the interactions between these VMs and isolate sets of VMs according to their purpose and domain membership.

The rationale for including these three references is that, being complementary, they provide a model for our VPH-Share contribution to secure the operations of the other Work Package 2 contributions, and, specifically, for cloud application secure deployment and execution.

In the context of cloud security, practical solutions and innovative models are being proposed at a fast pace. Among them, there is one with potentially far-reaching implications – once it becomes mature enough. Homomorphic encryption [205] is a scheme that allows computing arbitrary functions over encrypted data without a decryption key. With homomorphic encryption, sensitive data would always be encrypted, and could be handed over safely to cloud providers, who would then be able to process it without any need to "see" its contents. Successful solutions have been proposed [206], and optimization is ongoing to provide practical and mature results.

4.6.3.1 Security features of Eucalyptus

As Eucalyptus aims for full compatibility with Amazon Web Services (which, of course, involves using the same APIs, such as EC2 and S3), its security mechanisms are also similar. Depending on the applied tools, the following types of credentials may be used:

- ACCESS/SECRET key pair – two strings used to access Query (REST-based) interface e.g. using euca2ools;
- X.509 certificate and key – used by SOAP interface; e.g. in official ec2-* tools.

Both types of credentials can be obtained as a ZIP file from the Eucalyptus portal. The archive also contains the **eucarc** file to simplify credential usage. It seems that customization of credential backend is not straightforward in Eucalyptus.

In addition to authentication and authorization of all operations, Eucalyptus allows controlling access to predefined protocols/ports through **euca-authorize** as well as a mechanism allowing generation (with **euca-add-keypair**) and injection of RSA (or DSA) public keys into the instance, to allow passwordless authentication via the SSH protocol.

4.6.3.2 Security features of OpenNebula

The OpenNebula [32] cloud stack comes with an internal Authentication/Authorization module and the ability to use an external driver that takes care of these duties in its place. The internal module is based on OAuth, allows private/public key authentication and supports user quota, checking for user resource consumption before allowing a VM to be created in the system. OpenNebula is a very extensible and highly customizable system, and as such many add-ons can be found that extend its base security. An LDAP Authentication add-on centralizes authentication by allowing users to add their credentials to an external LDAP server. OpenNebula can be extended to encrypt the data it stores in its databases.

OpenNebula's unique feature of migrating VMs is protected with SSH, so as to establish a secure channel between physical machines.

When interconnecting VMs, OpenNebula is able to orchestrate several security technologies in order to combine both data centre resources and remote cloud resources, according to allocation policies.

4.6.3.3 Security features of OpenStack

As already mentioned, OpenStack is composed of two functionally distinct parts – computing part (Nova) and storage part (Swift).

Nova offers an Amazon-like API (much like Eucalyptus) and therefore operates on highly similar credentials (pairs of strings and X.509 certificate + private key). However, location, CA configuration and backend (driver) used for storing credentials can be easily customized [190] through **nova.conf**. At present, two drivers are supported – **nova.auth.dbdriver.DbDriver** (the default one – based on the main Nova database) and **nova.auth.ldapdriver.FakeLdapDriver** – which is just a "placeholder" that could be replaced (reimplemented) to provide a custom backend. Credentials can be downloaded as a ZIP archive using Nova CLI tools (**nova-manage**). This tool also facilitates other administration operations such as creating projects, assigning networks to projects etc.

OpenStack also supports **euca-authorize** and **euca-add-keypair** in the same way as Eucalyptus. In addition, it offers an interesting security feature called Cloudpipe [191] that allows creation of Virtual Private Networks (based on OpenVPN software) between any node external to the cloud and OpenStack project's private VLAN. Access is possible by using a certificate which is included in the ZIP archive and can be downloaded using CLI.

In Swift, authentication is handled through the so-called Auth service. It provides a RESTful API which has to be called with appropriate headers (X-Storage-User and X-Storage-Pass) to receive a special token (later used for authentication) and Storage URL. User credentials consist of two strings – account and username, separated by a colon (**:**). The password is a plain string. Creation of the initial (administrator) account requires the so-called **super_admin** key stored in **auth-server.conf**.

4.6.3.4 Security features of Nimbus

Nimbus' native (WSRF) API requires is strongly related to the Globus Toolkit and, as such, requires certificate and public key credentials. The same type of credentials may be used to access EC2 SOAP API. Additionally, Nimbus provides an EC2 Query API and its imaging service (Cumulus) S3 API – both APIs also use Amazon-like credentials, called *access id* and *access secret*. Users and their credentials are managed through a set of CLI tools (**nimbus-*-user**).

4.6.3.5 Security aspects of public Cloud Services

In this section we'd like to present the most important security-related features offered by public cloud providers:

- Amazon – the credentials used by this provider have already been mentioned in the document, as AWS APIs (EC2 and S3) are becoming a de-facto standard in the world of cloud stacks. Each access to AWS is either authenticated or anonymous. The AWS account entitles the user to possess an Access Key ID and a Secret Access Key, used by the REST Query API as well as a certificate and private key used by the SOAP API for authentication. Those credentials can be generated/accessed/deactivated/etc. through the AWS portal. Access to the portal is protected with e-mail (acting as a username) and password. Additionally, it is possible to enable the so-called AWS Multi-Factor Authentication [199]. In such cases, the user additionally has to provide a time-based One-Time Password generated by a special device to obtain access to the portal or the AWS Management Console. At present, MFA is also used for one specific API call – changing or deleting versioning state in the S3 bucket [192]. In addition to its authentication features, AWS offers a very powerful security mechanism called Amazon Virtual Private Cloud – which supports:
    - Creation of virtual private and public networks between instances;

  - Configuring access policies to the instances and S3 (restricting access to VPC defined IPs);
  - Running dedicated instances, i.e. instances run on hardware dedicated to a single client;
  - Creating IPsec-based VPNs with the ability to exchange routing information using the Border Gateway Protocol (BGP). Amazon offer generic instructions on configuration of these protocols, as well as specific ones for Cisco and Juniper routers [193]. There are also reports [194] that standard Linux server with IPsec tools (including tools for managing standard Linux 2.6.x IPsec stack and racoon daemon for IKE) and Quagga (for BGP) are sufficient to establish a VPN in Amazon.
- Rackspace – this provider's security features are definitely much less robust than in Amazon; Basic authentication mechanism is provided (both for CloudServers and CloudFiles) similar to OpenStack – the user needs to POST X-Auth-User and X-Auth-Key headers in the request to Auth server, and receives an X-Auth-Token along with additional information (such as URLs) needed for subsequent access to the service;
- GoGrid – provides similar authentication mechanisms to the ones used by AWS REST interfaces – the user must create an account (or accounts) choosing an arbitrary "shared secret" and "role" (e.g. super-user/read-only). In addition to these, the GoGrid portal generates an API key. All requests must be authorized with this API Key as well as an MD5 signature generated from the API Key, shared secret and timestamp. GoGrid also offers Dedicated Hardware Firewalls (Fortinet or Cisco) which can mitigate some network attacks as well as establish VPNs (including IPsec) to a private VLAN. GoGrid also offers the ability to use private VLANs to connect dedicated servers and cloud instances – yielding a hybrid solution. In this case, VPN may be used to access both types of services;
- Windows Azure – contrary to the already-mentioned providers, Azure is generally a PaaS, not an IaaS solution – with some exceptions. As, such it's natural that it offers more security-related features (being a higher-level platform). Azure uses various authentication mechanisms for accessing the management portal (which, in addition to typical management features, allows deployment of applications prepared as .cspkg packages). A LiveID account is used for direct deployment from Visual Studio – this exploits a pregenerated certificate. For storage access, user/password pairs are used. In addition to authentication features, Azure offers two security-related services:
  - Access Control [195] – allows identity and access control based both on MS Active Directory as well as various web solutions such as Live ID, and accounts provided by Google, Yahoo! or Facebook;
  - Connect [196] (currently still in technology preview stage) – allows creations of IPsec-based tunnels using IPv6 running on HTTPS [197]. According to Microsoft, this setup would provide secure communications between the client and the cloud, mitigating firewall-related issues through the use of HTTPS to encapsulate traffic.

4.6.3.6 OpenTC consortium

The OpenTC consortium (http://www.opentc.net/) focuses on developing an open trusted computing framework based on security mechanisms provided by low-level operating system layers with isolation properties and interfaces for trusted computing hardware. The OpenTC architecture enables security policy enforcement in the virtualisation layer [198] by defining and enforcing various kinds of security policies (ranging from access and flow policies to resource sharing policies).

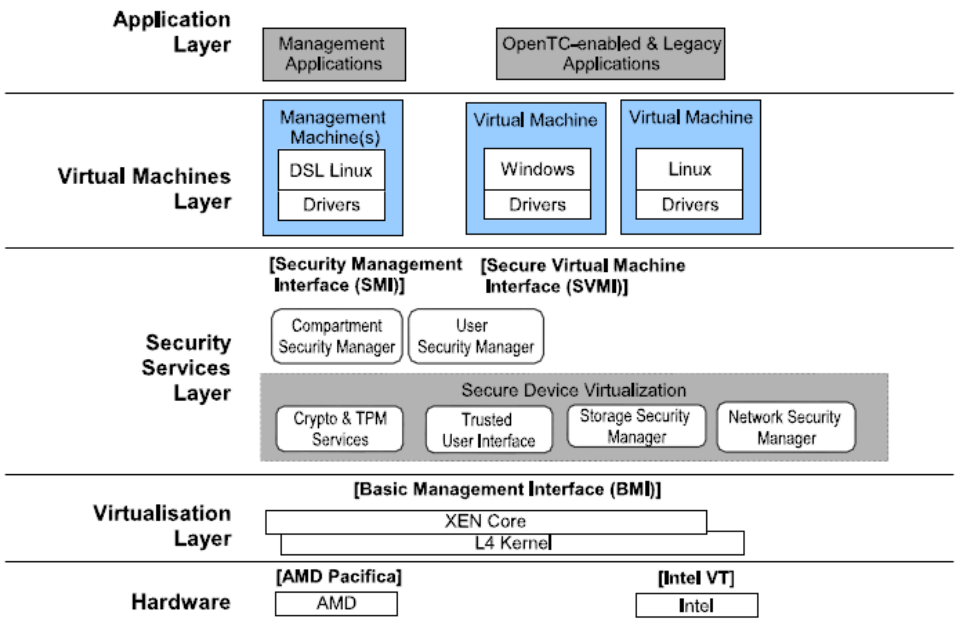

Figure 11: Layers of the OpenTC Architecture [198]

The Security Layer of this architecture offers a high flexibility due to the configurable policies, and also due to its Compartment Security Manager component. This component manages the life-cycle of the security policies associated to each “compartment” (i.e. VM). The Compartment Security Manager can be used to prove selected security properties to peers.

4.6.3.7 sHype

The sHype security architecture [199] was designed to provide policy-based access control for the shared virtual resources and the information flows between operating systems hosted on common hardware platforms and running on the same hypervisor.

sHype implements its security monitor in the hypervisor itself, to enforce resource sharing and information flow constraints between VMs. Access control is based on security policies, interpreted and enforced inside the hypervisor core, describing the access rights between VMs and virtual resources. Security labels are attached to VMs (subjects) and virtual resources (objects) to specify access requirements and authorisations. The sHype architecture supports various kinds of mandatory access control (MAC) policies, including Biba, Bell-LaPadula, Chinese Wall, Type Enforcement, and Caernarvon.

sHype follows the FLASK access control architecture [200], keeping the access control policy separate from access control enforcement. The sHype access control enforcement handles references of VMs at the hypervisor level and guards access to virtual resources according to the security policy. To make an access control decision, the Access Control Module applies access rules based on security information stored in security labels attached to VMs and virtual resources and the type of operation. The Access Control Module manages the definition of the security policies as well as the structure and interpretation of security labels for partitions and logical resources.

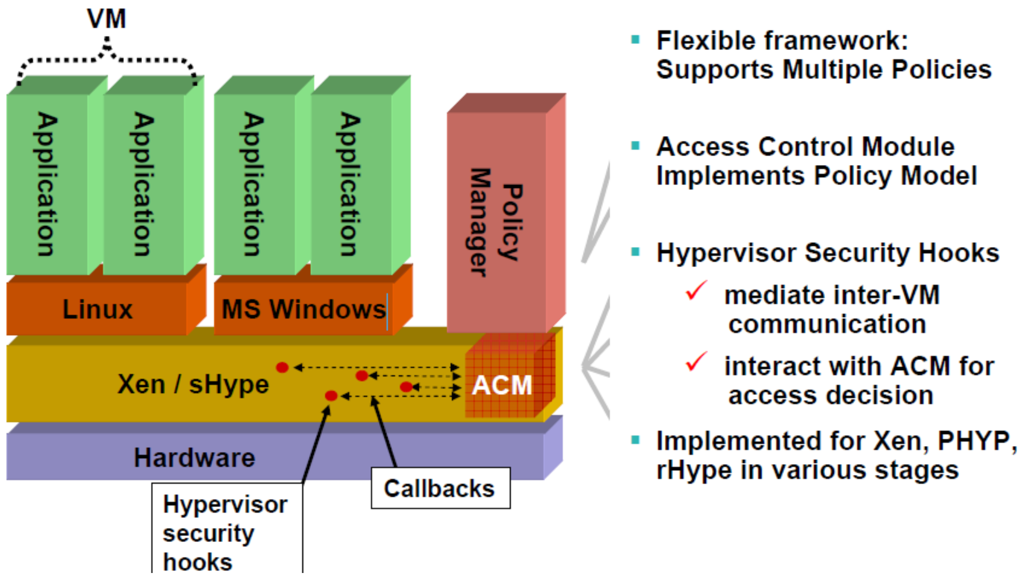
VM
Application
Application
Application
Application
Policy Manager
Linux
MS Windows
Xen / sHype
ACM
Hardware
Hypervisor security hooks
Callbacks
Flexible framework: Supports Multiple Policies
Access Control Module Implements Policy Model
Hypervisor Security Hooks
mediate inter-VM communication
interact with ACM for access decision
Implemented for Xen, PHYP, rHype in various stages

Figure 12: sHype architecture ACSAC05]

Hypervisors isolate the virtual resources, but do not control its sharing among VMs. The sHype architecture provides a policy-based system for controlling this information flow between VMs.

4.6.3.8 Trusted Virtual Domains

Trusted Virtual Domains [201] are sets of distributed Virtual Processing Elements (VPEs) in addition of the needed storage for these and the data exchange medium between them. A VPE is a set of VMs collaborating for a specific purpose, and isolated from other VPEs. The design of a TVD is based on the concept of Security Domains (separate computing environment that uniformly enforces a secure operational policy across all its members). This demands the application of isolation policies for aspects like storage, networking, and TVD membership.

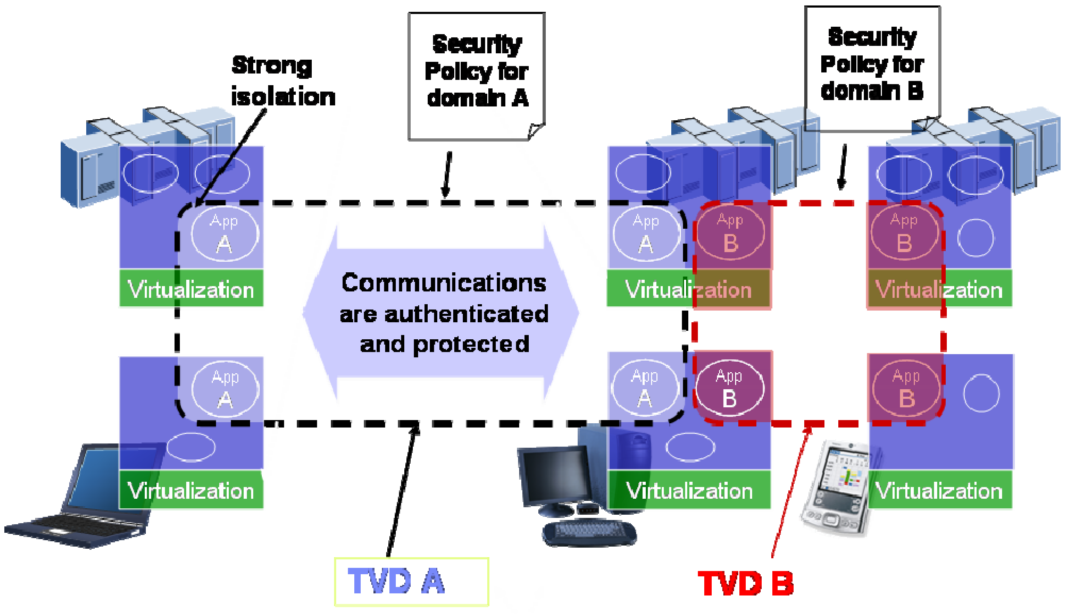
Strong isolation
Security Policy for domain A
Security Policy for domain B
App A
App B
Virtualization
Communications are authenticated and protected
TVD A
TVD B

Figure 13: Distributed services being performed in each TVD [201]

Within a TVD, high-level security and operational policy statements are mapped into the configuration of the individual hardware and software components that together perform a service. The policies go beyond the access control, and maintain the integrity of the TVD, as each member must prove the adherence to the security policy when joining.

Cabuk *et al.* [202] describe a framework based on TVD. It provides mechanisms to check the integrity of the Hypervisor executions, VMs operations and security policies enforcement in each virtual domain. This framework supports interactions between virtual domains (based on policies), including an integrity mechanism to allow trusted interactions between virtual domains by Trust Computing technology (e.g. secured IP-tunnels).

4.6.3.9 Related ongoing research projects

OPTIMIS (*Optimized Infrastructure Services*) [203] is a three-year research and development project selected under the “Software and Service Architectures & Infrastructures” track of the EU's FP7 framework program. OPTIMIS is aimed at enabling organizations to automatically externalize services and applications to trustworthy and auditable cloud providers in the cloud hybrid model. OPTIMIS aims to demonstrate cloud brokerage and federation across multiple cloud providers, as well as mechanisms by which organizations can scale out their infrastructure, using resources from third-party providers based upon a range of factors such as trust, risk assessment, eco-efficiency and cost. The OPTIMIS toolkit provides a set of components, covering functionalities needed by Infrastructure and Service Providers to build next generation cloud architectures. The OPTIMIS security framework is an integrated part of the OPTIMIS toolkit, and provides security functionalities within this scope.

The OPTIMIS security framework aims at applying Access Control, Identity Management and Security Policies administration to an Inter-Cloud scenario (Cloud Federation). A

Gateway acts as central point for all the security related services, with a Security Monitoring tool enabling the "secure multiple clouds" of the scenario by creating VPNs across different actors and resources.

The PASSIVE (*Policy-Assessed system-level Security of Sensitive Information processing in Virtualised Environments*) [204] EC-funded Project proposes an improved model of security for virtualised systems to ensure that:

- adequate separation of concerns (e.g. policing, judiciary) can be achieved even in large scale deployments;
- threats from co-hosted operating systems are detected and dealt with,
- public trust in application providers is maintained even in a hosting environment where the underlying infrastructure is highly dynamic.

The PASSIVE consortium proposes a Policy management system that enables fine-grained policies to be dynamically evaluated and enforced. It proposes two systems, one that handles the policies and another that implements it, thus keeping a separation between policy definition and policy enforcement.

To achieve these goals, the monitoring and access control will be built on a light weighted virtual machine manager. The project will focus as well on dynamically authenticating the applications and their support in a virtualized environment.

4.6.4 Restful Web service security for VPH-Share atomic services

Security for REST services is an open question. Instead of the standardization of SOAP web services (WS-* stack), REST services have only some best practices and models. There are no additional features on top of the HTTP application layer, and that means that the developers may only use standard HTTP mechanisms (such as HTTP authentication – basic or digest), HTTPS features (such as TLS mutual authentication) or create their own based on e.g. custom HTTP headers when composing the security for their services.

Nevertheless, and within the scope of cloud computing and VPH-Share requirements, there are some interesting models and solutions that we should review.

Amazon Web Services (AWS) provide an infrastructure web services platform in the cloud. They allow the management of computation and storage as the application demands them. In order to provide end-to-end security and end-to-end privacy, AWS builds services in accordance with security best practices and provides appropriate security features in those services. The way security is achieved in AWS presents a good model for any REST service in a cloud environment.

When requesting a service, you attach a signature to it. The Amazon service authenticates the request, and if successful, allows the invocation. Amazon charges for some actions (per-per-use pricing model), so having all your requests signed is necessary for the accountability of the services. The way Amazon authenticates the request is as follows: when signing the request, what actually happens is that it calculates the hash of the Secret Access Key provided when the account was created. This hash is attached to the request. Upon reception of the request, the Amazon platform is informed of which Secret Access Key the user 'claims' to have. It fetches it and repeats the composition of the signature (hash). If the two signatures (the one attached to the request and the one Amazon recreates to verify) match, then the Amazon platform considers the request to be authenticated and let's it go through. If they don't match, the request is rejected (and an error message sent to the "user"). The Amazon REST API validates also that the timestamp the request carries must be within 15 minutes of the Amazon system time, in order to prevent a malicious user to re-send a captured request again.

OAuth (Open Authorization) [200] is an open standard for authorization which allows the use of tokens for granting access to secured resources. It is widely used as a federated identity enabler, as it allows users to share protected resources stored in the Cloud with another user without having to hand out credentials. OAuth implementations don't provide token encrypting, so all communications should be via SSL.

Tokens are exchanged between requester and service (OAuth token negotiation protocol). Once approved, these OAuth tokens are stored in the client, thus serving as 'session identifiers', and are sent along with the REST request in the HTTP authorization header.

4.6.5 Recommendations for VPH-Share

As shown above various cloud stacks and providers use different types of credentials. As such we would need to provide integration mechanism between them.

It seems clear that some critical data needs to be processed in private cloud installations, or, at least, be pseudonymized prior to deployment to public clouds, to minimize potential harm if such data is compromised. During the design phase it might turn out that the project requires certain application-layer protocols that are not secured, but cannot be replaced (e.g. in legacy applications). In such cases, to ensure secure data transmission, it might be necessary to use one of the presented solutions which support creation of Virtual Private Networks between private and public clouds, clients and public clouds, or even specific cloud instances.

In VPH-Share, security requirements for atomic services make it necessary to secure RESTful services. As we have already stated, there is not a practical solution for REST services that can match the standardized security of SOAP services. We should consider two options to secure these REST services. Either we implement an authentication/authorization module for the services (which implies making the services dependent on this customized security for their operations), or, alternatively, we maintain secure messaging between these services and the WP2 tools using the SOAP WS-* stack of standard definitions (proxying their operations without any need to modify the services). This second option poses the challenge of transforming requests and responses from SOAP to REST and back.

Concerning authentication, we have seen in the State of the Art that most tools rely on available extensions which favour OAuth-based solutions. We recommend following this, and basing our authentication mechanisms on public/private keys.

5 Conclusions

The presented document should be treated as a stepping stone towards the development of a detailed WP2 architecture as well as selection of interfaces for other technical Work Packages of VPH-Share. The in-depth analysis of the state of the art, reported upon in this deliverable, enables us to make informed decisions regarding the tools, components and platforms which will be used when deciding upon the implementation and deployment specifics for each WP2 task.

The next three-month period of the project (Months 4-6) will be devoted to finalizing technical aspects of the WP2 architecture, based upon recommendations and requirements submitted by the workflow and user interface contributors via Task 2.7. At the end of Month 6 we expect to come up with a deliverable which will detail the features and internal workings of each component to be prepared by WP2 throughout the lifecycle of the Project. Having concluded this phase we will move on to preparing preliminary versions of WP2 tools and developing an integrated prototype cloud platform, which is expected to be produced by Month 12. We will also continue to liaise with workflow developers and affiliated projects to ensure that our tools match end-user expectations and can be used to productively expose VPH-Share application workflows in a distributed cloud environment.

6 References

[1] VPH-Share Grant Agreement for Collaborative Projec; Annex I – Description of Work (internal Consortium document).

[2] Jeffery K, Neidecker-Lutz B, The Future of Cloud Computing, Expert Group Report published by the European Commission Information Society and Media Directorate General – available at http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf (accessed May 9, 2011)

[3] The Apache HTTP Server Project; http://httpd.apache.org/

[4] HP Integrated Lights-Out (iLO) Advanced, http://h18013.www1.hp.com/products/servers/management/remotemgmt.html, HP, 2011

[5] DRAC: Dell Remote Access Card for Remote Server Management, http://www.dell.com/content/topics/global.aspx/power/en/ps2q02_bell?c=us&l=en, Dell, 2011

[6] Overview – Remote Supervisor Adapter II, http://www-947.ibm.com/support/entry/portal/docdisplay?lndocid=MIGR-50116, IBM, 2011

[7] Sun Integrated Lights Out Manager, http://www.sun.com/systemmanagement/ilom.jsp, Oracle, 2011

[8] Intel Remote Management Module (Intel RMM) – Overview, http://www.intel.com/Products/Server/Software/rmm/rmm-overview.htm, Intel, 2011

[9] Supermicro Intelligent Management, http://www.supermicro.com/products/nfo/IPMI.cfm, Supermicro, 2011

[10] Intelligent Platform Management Interface, http://www.intel.com/design/servers/ipmi/, Intel, 2011

[11] Preboot Execution Environment (PXE) Specification – Version 2.1, Intel, 1999; available at: http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf

[12] PXE Wiki, http://pxe.dev.aboveaverageurl.com/index.php/Main_Page, 2011

[13] Oracle Solaris Containers, http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html, Oracle, 2011

[14] Xen Hypervisor, http://www.xen.org/, Citrix Systems, 2011

[15] Kernel Based Virtual Machine, http://www.linux-kvm.org/page/Main_Page, 2011

[16] LXC Linux Containers, http://lxc.sourceforge.net/, 2011

[17] OpenVZ Wiki, http://wiki.openvz.org/Main_Page, 2011

[18] Virtualization, http://www.intel.com/technology/virtualization/, Intel, 2011

[19] AMD Virtualization (AMD-V) Technology, http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx, Advanced Micro Devices, 2011

[20] VMWare Workstations: http://www.vmware.com/products/workstation/

[21] RHEL6 Documentation – KVM limitations, http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization/sect-Virtualization-Virtualization_limitations-KVM_limitations.html, RedHat, 2011

[22] VMware ESXi and ESX Info Center, http://www.vmware.com/products/vsphere/esxi-and-esx/index.html, VMWare, 2011

[23] Hyper-V Server 2008 R2, http://www.microsoft.com/hyper-v-server/en/us/default.aspx, Microsoft, 2011

[24] Virtualization with Hyper-V: Supported Guest Operating Systems, http://www.microsoft.com/windowsserver2008/en/us/hyperv-supported-guest-os.aspx, Microsoft, 2011

[25] Microsoft Contributes Linux Drivers to Linux Community, http://www.microsoft.com/presspass/features/2009/Jul09/07-20LinuxQA.mspx, Microsoft, 2009

[26] OpenStack, http://www.openstack.org/, Rackspace, 2011

[27] Linux VServer Wiki, http://linux-vserver.org/Welcome_to_Linux-VServer.org, 2011

[28] OpenStack Announces Cactus Release, http://www.openstack.org/blog/2011/04/openstack-announces-cactus-release/, Rackspace, 2011

[29] FreeBSD Wiki – Jails, http://wiki.freebsd.org/Jails, 2011

[30] Debian GNU/kFreeBSD inside native FreeBSD jail, http://phaq.phunsites.net/2007/01/06/debian-gnukfreebsd-inside-native-freebsd-jail/, 2007

[31] Eucalyptus, http://open.eucalyptus.com/, Eucalyptus Systems, 2011

[32] OpenNebula, http://opennebula.org/, OpenNebula Project Leads, 2011

[33] Nimbus, http://www.nimbusproject.org/, University of Chicago, 2011

[34] Full OpenVZ support for Nova, https://blueprints.launchpad.net/nova/+spec/openvz-driver, 2011

[35] Heterogeneous Architecture Scheduler, http://wiki.openstack.org/HeterogeneousArchitectureScheduler, 2011

[36] PC_scheduler, http://wiki.openstack.org/PC_scheduler, 2011

[37] RackSpace CloudFiles; http://www.rackspace.com/cloud/cloud_hosting_products/files/

[38] OpenStack Swift Documentation – Sample Installation Architecture, http://docs.openstack.org/cactus/openstack-object-storage/admin/content/example-installation-architecture-swift.html, 2011

[39] OpenStack Swift Documentation – System Requirements, http://docs.openstack.org/cactus/openstack-object-storage/admin/content/object-storage-system-requirements.html, 2011

[40] Swift3, http://swift.openstack.org/misc.html#module-swift.common.middleware.swift3, 2011

[41] The Top 150 Players in Cloud Computing, http://cloudcomputing.sys-con.com/node/770174, Cloud Computing Journal, 2009

[42] Li A, Yang X, Kandula S, and Zhang M, CloudCmp: Comparing Public Cloud Providers, Association for Computing Machinery, Inc., 2010; available at: http://research.microsoft.com/apps/pubs/?id=136448

[43] AWS in Education, http://aws.amazon.com/education/, Amazon, 2011

[44] Amazon Web Services, http://aws.amazon.com/, Amazon, 2011

[45] RackSpace Cloud, http://www.rackspace.com/cloud/, Rackspace, 2011

[46] Google App Engine, http://code.google.com/intl/en/appengine/, Google, 2011

[47] Microsoft Azure storage services; http://www.microsoft.com/windowsazure/storage/ (accessed May 9, 2011)

[48] Salesforce.com, http://www.salesforce.com/eu/?ir=1, Salesforce.com, 2011

[49] Heroku, http://www.heroku.com/, 2011

[50] PiCloud, http://www.picloud.com/, PiCloud, 2011

[51] GoGrid, http://www.gogrid.com/, GoGrid, 2011

[52] Science Clouds, http://scienceclouds.org/, University of Chicago, 2011

[53] FutureGrid, https://portal.futuregrid.org/, 2011

[54] Shubert L at al., The Future Of Cloud Computing – Opportunities For European Cloud Computing Beyond 2010, European Commission; 2011

[55] Amazon Virtual Private Cloud (Amazon VPC), http://aws.amazon.com/vpc/, Amazon, 2011

[56] Rubin E, 2010 is the Year of the Federated Cloud, CloudSwitch, 2010; available at: http://www.cloudswitch.com/page/2010-is-the-year-of-the-federated-cloud

[57] Federated Clouds? Possible? http://www.virtualizationpractice.com/blog/?p=10481, 2011

[58] Open Cloud Computing Interface, http://occi-wg.org/, Open Grid Forum, 2011

[59] Deltacloud, http://incubator.apache.org/deltacloud/, 2011

[60] Right_aws, http://rightaws.rubyforge.org/, RightScale, 2011

[61] Buyya R, Yeo CS, Venugopal S. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities, Aug. 2008. [Online]. Available: http://arxiv.org/abs/0808.3558

[62] Vaquero LM, Merino LR, Caceres J, Lindner M, A Break in the Clouds: Towards a Cloud Definition, SIGCOMM Comput. Commun. Rev., vol. 39, no. 1, pp. 50-55, Dec. 2008. [Online]. Available: http://dx.doi.org/10.1145/1496091.1496100

[63] Armbrust M, Fox A, Grifth R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report, University of California at Berkeley, February 2009.

[64] Campbell S, Security – The Dark Side of the Cloud, January 25, 2010; http://www.hpcwire.com/hpcwire/2010-01-25/security_--_the_dark_side_of_the_cloud.html

[65] Pearson S, Taking Account of Privacy when Designing Cloud Computing, HP Laboratories, March 6, 2009

[66] CloudHarmony Blog: An unofficial EC2 Outage Postmortem – The Sky is not Falling; http://blog.cloudharmony.com/2011/04/unofficial-ec2-outage-postmortem-sky-is.html, April 25, 2011

[67] Web service architecture. http://www.w3.org/TR/ws-arch/

[68] RESTful Web Services: The Basics. https://www.ibm.com/developerworks/webservices/library/ws-restful/

[69] The SOAPLab project and libraries. http://soaplab.sourceforge.net/soaplab2/

[70] Kandaswamy G, Gannon D. A mechanism for Just-In-Time Creation of Web Services for Scientific Workflows. Workshop on Web Services-based Grid Applications, August 2006

[71] Apache Commons Exec library. http://commons.apache.org/exec/

[72] Ruby programming language. http://www.ruby-lang.org/

[73] Python programming language. http://www.python.org/

[74] Apache CXF: An Open-Source Services Framework. http://cxf.apache.org/

[75] Python web service library. http://pywebsvcs.sourceforge.net/

[76] Porter G, Katz RH, Effective Web Service Load Balancing through Statistical Monitoring, in Communications of the ACM 49 (3), 48-54, March 2006

[77] Hollingsworth J, Tierney B. Monitoring and Instrumentation, In: I. Foster and C. Kesselman, eds., The Grid 2: Blueprint for a New Computing Infrastructure, pages 319–351. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[78] Baliś B, Monitoring of Grid Scientific Workflows, Ph.D. Thesis, AGH University of Science and Technology, November 2008.

[79] de Rose LA, Reed DA. SvPablo: A Multi-Language Architecture-Independent Performance Analysis System. In ICPP ’99: Proceedings of the 1999 International Conference on Parallel Processing, page 311, Washington, DC, USA, 1999. IEEE Computer Society.

[80] Cargille J, Miller BP. Binary Wrapping: a Technique for Instrumenting Object Code. SIGPLAN Not., 27(6):17–18, 1992.

[81] Larus JR, Schnarr E, EEL: Machine-Independent Executable Editing. ACM SIGPLAN Not., 30(6):291–300, 1995.

[82] Buck B, Hollingsworth JK, An API for Runtime Code Patching, Int. J. High Perform. Comput. Appl., 14(4):317–329, 2000.

[83] Yigitbasi N, Iosup A, Epema D, Ostermann S, C-Meter: A Framework for Performance Analysis of Computing Clouds. Volume 0, Los Alamitos, CA, USA, pp. 472-477. IEEE Computer Society 2009

[84] Juve G, Deelman E, Vahi K, Mehta G, Scientific Workflow Applications on Amazon EC2, Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE International Conference on e-Science (e-Science 2009), Oxford, UK: 2009.

[85] Walker E, Benchmarking Amazon EC2 for High-Performance Scientific Computing, LOGIN 33 (5), October 2008, 18-23.

[86] http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html

[87] The CloudExchange portal; http://cloudexchange.org.

[88] Singh G, Kesselman C, Deelman E, A Provisioning Model and its Comparison with Best-Effort for Performance-Cost Optimization in Grids, in Proceedings of the 16th international symposium on High performance distributed computing, ser. HPDC '07. New York, NY, USA: ACM, 2007, pp. 117-126. [Online]. Available: http://dx.doi.org/10.1145/1272366.1272382

[89] Fonseca CM, Fleming PJ, Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion, and Generalization, in: Proceedings of the Fifth International Conference on Genetic Algorithms. 1993.

[90] Pandey S, Barker A, Gupta KK, Buyya R, Minimizing Execution Costs when using Globally Distributed Cloud Services, 2010 24th IEEE International Conference on Advanced Information Networking and Applications

[91] Fourer R, Gay DM, Kernighan BW. AMPL: A Modeling Language for Mathematical Programming. Duxbury Press, November 2002.

[92] Spellucci P. A SQP Method for General Nonlinear Programs using Only Equality Constrained Subproblems. Mathematical Programming, 82:413–448, 1993.

[93] Chandra A, Gong W, Shenoy P, Dynamic Resource Allocation for Shared Data Centers using Online Measurements, In: SIGMETRICS ’03: Proceedings of the 2003 ACM SIGMETRICS international Conference on Measurement and Modeling of Computer systems. ACM Press, New York, NY, USA, pp. 300–301.

[94] Parekh AK, Gallager RG, A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Multiple Node Case, in Networking, IEEE/ACM Transactions on 2 (2), 137–150, 1994

[95] Van HN, Tran FD, and Menaud J-M, SLA-Aware Virtual Resource Management for Cloud Infrastructures, in: CIT ’09: Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology Washington, DC, USA: IEEE Computer Society, 2009, pp. 357–362. [Online]. Available: http://dx.doi.org/http://dx.doi.org/10.1109/CIT.2009.109

[96] Rochwerger B, Breitgand D, Levy E, Galis A, Nagin K, Llorente IM, Montero R, Wolfsthal Y, Elmroth E, Cáceres J, Ben-Yehuda M, Emmerich W, Galán F, The RESERVOIR model and Architecture for Open Federated Cloud Computing, IBM J. Res. Dev. 53, 4 (July 2009), 535-545.

[97] Sotomayor B et al., Capacity Leasing in Cloud Systems using the OpenNebula Engine. Cloud Computing and Applications 2008 (CCA08).

[98] Marshall P, Keahey K, Freeman T, Improving Utilization of Infrastructure Clouds, Available online: http://www.mcs.anl.gov/uploads/cels/papers/P1845.pdf

[99] Kephart JO, Chess D, The Vision of Autonomic Computing, Computer, vol. 36, no. 1, 2003, pp. 41-50.

[100] Menasce DA, Kephart JO. Guest Editors' Introduction: Autonomic Computing. IEEE Internet Computing. 2007;11(1):18-21. Available from: http://dx.doi.org/10.1109/MIC.2007.11.

[101] Bennani M, Menasce DA, Resource Allocation for Autonomic Data Centers Using Analytic Performance Models, Proceedings of 2nd International Conference on Autonomic Computing, ICAC 05, IEEE CS Press, 2005, pp. 229-240.

[102] Walsh WE et al., Utility Function in Autonomic Computing, Proceedings of 1st International Conference on Autonomic Computing, ICAC 04, IEEE CS Press, 2004, pp. 70-77.

[103] Menasce DA, Dodge R, Barbara D, Preserving QoS of E-Commerce Sites through Self-Tuning: A Performance Model Approach, Proceedings of 2001 ACM Conference on E-Commerce, ACM Press, 2001.

[104] Diao Y et al., Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Server, in Proceedings of IEEE/IFIP Network Operations and Management Symposium, IEEE CS Presss, 2002, pp. 219-234.

[105] Kephart JO, Research Challenges of Autonomic Computing, Proceedings of International Conference on Software Engineering, ACM Press, 2005, pp. 15-22.

[106] Buyya R, Murshed M (2002). GridSim: a Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience, 14 (13-15), 1175-1220.

[107] Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2011). CloudSim: a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Softw. Pract. Exper. 41 (1), 23-50.

[108] Ostermann S, Plankensteiner K, Prodan R, Fahringer T (August 2010). GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds. In: CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing. Springer.

[109] Woollard D, Mattmann C, Medvidovic N, Injecting Software Architectural Constraints into Legacy Scientific Applications, In Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering IEEE Computer Society Washington, DC, USA 2009

[110] Distributed European Infrastructure for Supercomputing Applications (DEISA): http://www.deisa.org (accessed May 9, 2011)

[111] The Partnership for Advanced Computing in Europe (PRACE): http://www.prace-project.eu/ (accessed April 25, 2011)

[112] Coveney PJ, Scientific Grid computing. Phil. Trans. R. Soc. A, 2005. 363: p. 1701-2095.

[113] Foster I, Globus Toolkit Version 4: Software for Service-Oriented Systems. Journal of Computer Science and Technology, 2006. 21(513).

[114] The UNICORE Project: www.unicore.eu (accessed May 9, 2011)

[115] gLite Middleware: http://glite.web.cern.ch/glite/ (accessed May 9, 2011)

[116] Boghosian B et al., NEKTAR, SPICE and Vortonics: Using Federated Grids for Large Scale Scientific Applications. Cluster Computing, 2007.

[117] Chin J, Coveney PJ, Towards tractable toolkits for the grid: A plea for lightweight, useable middleware, Technical report, 2004.

[118] Zasada SJ, Coveney PJ, Virtualizing Access to Scientific Applications with the Application Hosting Environment, Computer Physics Communications, 2009, 180: p. 2513-2525.

[119] Hayes M et al., GROWL: A Lightweight Grid Services Toolkit and Applications. UK e-Science All Hands Meeting, 2007.

[120] van Nieuwpoort RV, Kielmann T, Bal HE, User-Friendly and Reliable Grid Computing Based on Imperfect Middleware. in ACM/IEEE conference on Supercomputing. 2007.

[121] Blower JD, Harrison AB, Haines K. Styx Grid Services: Lightweight, easy-to-use middleware for scientific workflows. in Proceedings of the 4th UK e-Science All Hands Meeting; 2005.

[122] Graham S et al., Web services resource framework, 2006.

[123] WSRF::Lite: http://www.sve.man.ac.uk/research/AtoZ/ILCT (accessed May 9, 2011)

[124] SOAP::Lite Web Services Toolkit: http://www.soaplite.com (accessed May 9, 2011)

[125] Grid Job Submission and Monitoring Web Service: http://gridsam.sorceforge.net (accessed May 9, 2011)

[126] Sun Grid Engine: http://gridengine.sunsource.net (accessed May 9, 2011)

[127] Pickles SM et al., A Practical Toolkit for Computational Steering. Phil. Trans. R. Soc. A, 2005. 363: p. 1843-1853.

[128] MacLaren J, Keown M, Pickles SM, Co-allocation, Fault Tolerance and Grid Computing; in: Proceedings of the UK e-Science All Hands Meeting, 2006.

[129] Lamport L, Paxos made Simple. SIGACTN: SIGACT News (ACM special Interest Group on Automata and Computability Theory), 2001. 32: p. 18-25.

[130] Coveney PJ et al., Large scale computational science on federated international grids: The role of switched optical networks. Future Generation Computer System, 2007.

[131] Job Submission Description Language Specification 21.

[132] Haidar AN et al., Audited Credential Delegation: A Usable Security Solution for the Virtual Physiological Human Toolkit. Interface Focus, 2011.

[133] Zasada SJ, Haidar AN, Coveney PJ, On the Usability of Grid Middleware and Security Mechanisms. Philosophical Transactions of the Royal Society A, 2010a.

[134] V., W. X. 509 Proxy Certificates for Dynamic Delegation. in 3rd Annual PKI R&D Workshop, 2004.

[135] Martin A, Spencer D, Trust and Security in Virtual Communities, Report on First Workshop: the Application-Led Security Agenda for e-Science, 2008.

[136] Beckles B, Welch V, Basney J, Mechanisms for increasing the usability of grid security. Int. J. Human-Computer Studies, 2005. 63: p. 74-101.

[137] Kuno H et al., Web Services: Concepts, Architectures and Applications 2004: Springer.

[138] Abdallah, AE, Khayat EJ. Formal Z Specifications of Several Flat Role-Based Access Control Models. in In 30th Annual IEEE/NASA Software Engineering Workshop. 2006.

[139] The MyProxy credential management service: http://grid.ncsa.uiuc.edu/myproxy (accessed May 9, 2011)

[140] Hull D et al., Taverna: a tool for building and running workflows of services. Nucleic Acids Research, 2006. 34: p. 729-732.

[141] Deelman E et al., Workflows and e-Science: An overview of workflow system features and capabilities. Future Generation Computer Systems, 2008.

[142] Allen G et al. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid; in Proceedings of the IEEE.

[143] Amin K et al., An abstraction model for a grid execution framework. Journal of Systems Architecture. The EUROMICRO Journal, 2006. 52.

[144] Goodale T et al., SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. Computational Methods in Science and Technology, 2006. 12: p. 7-20.

[145] Distributed Resource Management Application API Specification 1.0 (DRMAA): http://www.drmaa.org (accessed May 9, 2011)

[146] Nakada H et al., The GridRPC API standardization at the 2005. GGF proposed recommendation, in: Grid Remote Procedure Call Working Group of the Global Grid Forum. 2005.

[147] The Virolab Project: http://www.virolab.org/ (accessed May April 25, 2011)

[148] Kalé L et al., NAMD2: Greater Scalability for Parallel Molecular Dynamics. Journal of Computational Physics, 1999. 151(1): p. 283-312

[149] Brooks B et al., CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. Journal of Computational Chemistry, 1983. 4(2): p. 187-217.

[150] Plimpton SJ, Fast Parallel Algorithms for Short-Range Molecular Dynamics. J Comp Phys, 1995. 117: p. 1-19.

[151] Sun G et al., Performance of the Vienna ab initio Simulation Package (VASP) in Chemical Applications, Journal of Molecular Structure: THEOCHEM, 2003. 624(1): p. 37-45.

[152] Smith W, Forester TR, DL_POLY_2.0: A General-Purpose Parallel Molecular Dynamics Simulation Package. Journal of Molecular Graphics, 1999. 14(3).

[153] Saksena RS, Coveney PJ, Self-Assembly of Ternary Cubic, Hexagonal, and Lamellar Mesophases using the Lattice-Boltzmann Kinetic Method, J. Phys. Chem. B., 2008. 112: p. 2950-2957.

[154] Mazzeo MD, Coveney PJ, HemeLB: A High-Performance Parallel Lattice-Boltzmann Code for Large Scale Fluid Flow in Complex Geometries. Computer Physics Communications, 2008. 178: p. 894-914.

[155] Buyya R, Ranjan R, Calheiros R, Intercloud: Utility-oriented Federation of Cloud Computing Environments for Scaling of Application Services. In: Ching-Hsien H, Yang L,

Park P, Sang-Soo Y, eds., Algorithms and Architectures for Parallel Processing, volume 6081 of Lecture Notes in Computer Science, pages 13–31. Springer Berlin/Heidelberg, 2010.

[156] Elmroth E, Marquez FG, Henriksson D, Ferrera DP. Accounting and Billing for Federated Cloud Infrastructures, International Conference on Grid and Cloud Computing, 0:268–275, 2009.

[157] Celesti A, Tusa F, Villari M, Puliafito A, How to Enhance Cloud Architectures to enable Cross-Federation, IEEE International Conference on Cloud Computing, 0:337–345, 2010.

[158] Celesti A, Tusa F, Villari M, Puliafito A, Improving Virtual Machine Migration in Federated Cloud Environments. International Conference on the Evolving Internet, 0:61–67, 2010.

[159] The Cloud Storage Broker; http://www.oxygencloud.com/cloud-storage-broker

[160] Vukolić M, The Byzantine Empire in the Intercloud, SIGACT News, 41:105–111, September 2010.

[161] Cachin C, Haas R, Vukolić M, Dependable Storage in the Intercloud, Aug 2010.

[162] Graham-Rowe D, A Faster Way to the Cloud. http://www.technologyreview.com/computing/23451/

[163] Gu Y, Grossman RL, UDT: UDP-Based Data Transfer for High-Speed Wide Area Networks, Comput. Netw., 51:1777–1799, May 2007.

[164] Gu Y, Grossman RL, Sector and Sphere: The Design and Implementation of a High Performance Data Cloud.

[165] Yang Y, Zhou Y, Liang L, He D, Sun Z, A Service-Oriented Broker for Bulk Data Transfer in Cloud Computing, in: GCC'10, pp. 264–269.

[166] The CloudStandards website; http://cloud-standards.org

[167] Open Grid Forum. http://www.gridforum.org/

[168] Jclouds: http://www.jclouds.org/

[169] OCCI4Java: https://github.com/occi4java/occi4java/

[170] Mellor C, Swissdisk Suffers Spectacular Cloud Snafu, The Register 2009; published at http://www.theregister.co.uk/2009/10/19/swissdisk_failure/ (accessed May 5, 2011)

[171] Leung AW, Pasupathy S; Goodson G, Miller EL, Measurement and analysis of large-scale network file system workloads, ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference, USENIX Association, p. 213-226

[172] Amazon Simple Storage Service FAQ; http://aws.amazon.com/s3/faqs/#How_reliable_is_Amazon_S3 (accessed May 9, 2011)

[173] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W, Dynamo: Amazon's Highly Available Key-Value Store, SOSP2007 – available at: http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf (accessed May 9, 2011)

[174] RackSpace Dedicated Network Attached Storage; http://www.rackspace.com/managed_hosting/services/storage/dnas.php (accessed May 9, 2011)

[175] GoGrid Cloud Storage, http://www.gogrid.com/cloud-hosting/cloud-storage.php (accessed May 9, 2011)

[176] Juels A, Kaliski BS, Pors: proofs of retrievability for large files. In Proceedings of the 14th ACM conference on Computer and communications security, CCS '07, New York, NY, USA, pp. 584-597. ACM. http://www.citeulike.org/user/malawski/article/9245722; http://dx.doi.org/10.1145/1315245.1315317 (accessed May 2, 2011)

[177] Bowers KD, Juels A, Oprea A, HAIL: a high-availability and integrity layer for cloud storage. In Proceedings of the 16th ACM conference on Computer and communications security, CCS '09, New York, NY, USA, pp. 187-198. ACM. http://www.citeulike.org/user/malawski/article/6938035; http://dx.doi.org/10.1145/1653662.1653686 (accessed May 2, 2011)

[178] Shraer A, Cachin C, Cidon A, Keidar I, Michalevsky Y, Shaket D (2010); Venus: Verification for Untrusted Cloud Storage. In: Proceedings of the 2010 ACM workshop on Cloud computing security workshop, CCSW '10, New York, NY, USA, pp. 19-30. ACM. http://www.citeulike.org/user/malawski/article/9245722; http://dx.doi.org/10.1145/1315245.1315317 (accessed May 2, 2011)

[179] Kent S, Seo K, Request for Comments: 4301 – Security Architecture for the Internet Protocol, 2005

[180] Kent S, Request for Comments: 4302 – IP Authentication Header, 2005

[181] Kent S, Request for Comments: 4303 – IP Encapsulating Security Payload (ESP), 2005

[182] Kaufman C et al., Request for Comments: 5996 – Internet Key Exchange Protocol Version 2 (IKEv2), 2010

[183] Lau J, Townsley M, Goyret I, Request for Comments: 3931 – Layer Two Tunneling Protocol - Version 3 (L2TPv3), 2005

[184] OpenVPN Community Wiki and Tracker, https://community.openvpn.net/openvpn (accessed May 27, 2011)

[185] Microsoft, [MS-IPHTTPS]: IP over HTTPS (IP-HTTPS) Tunneling Protocol Specification - http://msdn.microsoft.com/en-us/library/dd358571%28v=PROT.10%29.aspx (accessed May 27, 2011)

[186] Fail2Ban, http://www.fail2ban.org/wiki/index.php/Main_Page (accessed May 27, 2011)

[187] Snort, http://www.snort.org/ (accessed May 27, 2011)

[188] Nessus, http://www.tenable.com/products/nessus (accessed May 27, 2011)

[189] OpenVAS, http://www.openvas.org/ (accessed May 27, 2011)

[190] OpenStack – Configuring Authentication and Authorization, http://docs.openstack.org/cactus/openstack-compute/admin/content/configuring-authentication-authorization.html (accessed May 27, 2011)

[191] OpenStack – Cloudpipe, http://docs.openstack.org/cactus/openstack-compute/admin/content/cloudpipe-per-project-vpns.html (accessed May 27, 2011)

[192] Amazon AWS – Configuring a Bucket with MFA Delete, http://docs.amazonwebservices.com/AmazonS3/latest/dev/index.html?ConfiguringaBucketwithMFADelete.html (accessed May 27, 2011)

[193] Amazon Virtual Private Cloud – Network Administrator Guide - http://docs.amazonwebservices.com/AmazonVPC/2009-07-15/NetworkAdminGuide/ (accessed May 27, 2011)

[194] Amazon VPC with Linux – http://openfoo.org/blog/amazon_vpc_with_linux.html (accessed May 27, 2011)

[195] Microsoft Azure – Access Control http://www.microsoft.com/windowsazure/appfabric/accesscontrol/ (accessed May 27, 2011)

[196] Overview of Windows Azure Connect – http://msdn.microsoft.com/en-us/library/gg432997.aspx (accessed May 27, 2011)

[197] Overview of Firewall Settings Related to Windows Azure Connect, http://msdn.microsoft.com/en-us/library/gg433061.aspx (accessed May 27, 2011)

[198] Kuhlmann D, Landfermann R, Ramasamy HV, Schunter M, Ramunno G, Vernizzi D, An Open Trusted Computing Architecture – Secure Virtual Machines Enabling User-Defined Policy Enforcement, OpenTC report, 2006

[199] Sailer R, Valdez E, Jaeger T, Perez R, van Doorn L, Griffin JL, Berger S, sHype: Secure Hypervisor Approach to Trusted Virtualized Systems, IBM Research Report (RC23511), 2005

[200] Spencer R, Smalley S, Loscocco P, Hibler M, Andersen D, Lepreau J, The Flask Security Architecture: System Support for Diverse Security Policies, Proceedings of the 8th conference on USENIX Security Symposium – Volume 8, 1999

[201] Bussani A, Griffin JL, Jansen B, Julisch K, Karjoth G, Maruyama H, Nakamura M, Perez R, Schunter M, Tanner A, van Doorn L, Herreweghen EV, Waidner M, Yoshihama S, Trusted Virtual Domains: Secure Foundation for Business and IT Services, Research Report RC 23792, IBM Research, November 2005.

[202] Cabuk S, Dalton CI, Eriksson K, Kuhlmann D, Ramasamy HV, Ramunno G, Sadeghi AR, Schunter M, Stüble C, Towards Automated Security Policy Enforcement in Multi-tenant Virtual Data Centers, Journal of Computer Security, Volume 18, Number 1, 2010

[203] Optimized Infrastructure Services, http://www.optimis-project.eu, 2011

[204] Policy-Assessed System-level Security of Sensitive Information processing in Virtualised Environments, http://ict-passive.eu, 2011

[205] Micciancio D, A First Glimpse of Cryptography's Holy Grail. Association for Computing Machinery. p. 96.

[206] Gentry C, A fully Homomorphic Encryption Scheme, Dissertation submitted to the Department of Computer Sciences and the Committee on Graduate Studies of Stanford University, http://crypto.stanford.edu/craig/craig-thesis.pdf, 2009

[207] AWS Multi-Factor Authentication, http://aws.amazon.com/mfa/, 2011

[208] The Internet Engineering Task Force (IETF), The OAuth 2.0 Protocol, 2010

List of Key Words/Abbreviations

API	Application Programmer’s Interface
AWS	Amazon Web Services
BLOB	Binary Large Object
CCU	Cloud Compute Unit
CLI	Command-Line Interface
DHCP	Dynamic Host Configuration Protocol
DNAS	Dynamic Network Authentication System
DRAC	Dell Remote Access Card
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
FASP	Fast And Secure Protocol
FC	Fibre Channel
FLASK	Flux Advanced Security Kernel
GFS2	Global File System 2
GSI	Globus Security Infrastructure
HAIL	High Availability and Integrity Layer
HPC	High-Performance Computing
IaaS	Infrastructure as a Service
iLO	Integrated Lights-Out
ILOM	Sun Integrated Lights Out Manager
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
iSCSI	Internet Small Computer Systems Interface
KVM	Kernel-based Virtual Machine

LAN	Local Area Network
LXC	Linux Containers
MAC	Media Access Control
NAS	Network Attached Storage
NFS	Network File System
OASIS	Organization for the Advancement of Structured Information Standards
OCCI	Open Cloud Computing Interface
OGF	Open Grid Forum
PaaS	Platform as a Service
PDU	Power Distribution Unit
POR	Proof Of Retrievability
PXE	Preboot Execution Enviroment
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RDBMS	Relational Database Management System
REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
RMM	Remote Management Module
RSA	Remote Supervisor Adapter
RTT	Real Time Transfer
S3	Simple Storage Service
SaaS	Software as a Service
SAN	Storage Area Network
SDK	Software Development Kit
SLA	Service Level Agreement

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRB	Storage Resource Broker
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	Universal Datagram Protocol
UDT	UDP-Based Data Transfer
VHD	Virtual Hard Disk
VM	Virtual Machine
VMM	Virtual Machine Monitor (Hypervisor)
WAN	Wide Area Network
VENUS	Verification for Untrusted Storage
WSRF	Web Services Resource Framework