Flexible services for the support of research

Author(s): Matteo Turilli, David Wallom, Chris Williams, Steve Gough, Neal Curran, Richard Tarrant, Dan Bretherton, Andy Powell, Matt Johnson, Terry Harmer, Peter Wright and John Gordon

Source: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 28 January 2013, Vol. 371, No. 1983, e-Science-towards the cloud: infrastructures, applications and research (28 January 2013), pp. 1-20

Published by: Royal Society

Stable URL: https://www.jstor.org/stable/41739957

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A

rsta.royalsocietypublishing.org

Research

Cite this article: Turilli M, Wallom D, Williams C, Gough S, Curran N, Tarrant R, Bretherton D, Powell A, Johnson M, Harmer T, Wright P, Gordon J. 2013 Flexible services for the support of research. Phil Trans R Soc A 371: 20120067.
http://dx.doi.org/10.1098/rsta.2012.0067

One contribution of 13 to a Theme Issue 'e-Science–towards the cloud: infrastructures, applications and research'.

Subject Areas:
e-Science, software

Keywords:
cloud computing, infrastructure as a service, cloud federation, cloud storage

Authors for correspondence:
Matteo Turilli
e-mail: matteo.turilli@oerc.ox.ac.uk
David Wallom
e-mail: david.wallom@oerc.ox.ac.uk

†These authors contributed equally to this study.

Royal Society Publishing
Informing the science of the future

Flexible services for the support of research

Matteo Turilli[1,†], David Wallom[1,†], Chris Williams[1], Steve Gough[2], Neal Curran[2], Richard Tarrant[2], Dan Bretherton[2], Andy Powell[3], Matt Johnson[3], Terry Harmer[4], Peter Wright[4] and John Gordon[5]

[1]Oxford e-Research Centre, University of Oxford, Oxford OX1 3QG, UK
[2]Information Services, University of Reading, Reading RG6 6AF, UK
[3]Eduserv, Royal Mead, Railway Place, Bath BA1 1SR, UK
[4]Eoverl Ltd, 3 Wellington Park, Belfast BT9 6DJ, UK
[5]Science & Technology Facilities Council Rutherford Appleton Laboratory, Harwell, Oxford, Didcot OX11 0QX, UK

Cloud computing has been increasingly adopted by users and providers to promote a flexible, scalable and tailored access to computing resources. Nonetheless, the consolidation of this paradigm has uncovered some of its limitations. Initially devised by corporations with direct control over large amounts of computational resources, cloud computing is now being endorsed by organizations with limited resources or with a more articulated, less direct control over these resources. The challenge for these organizations is to leverage the benefits of cloud computing while dealing with limited and often widely distributed computing resources. This study focuses on the adoption of cloud computing by higher education institutions and addresses two main issues: flexible and on-demand access to a large amount of storage resources, and scalability across a heterogeneous set of cloud infrastructures. The proposed solutions leverage a federated approach to cloud resources in which users access multiple and largely independent cloud infrastructures through a highly customizable broker layer. This approach allows for a uniform authentication and authorization infrastructure, a fine-grained policy specification and the aggregation of accounting and monitoring. Within a loosely coupled federation of cloud infrastructures, users can access vast amount of data without copying them across cloud infrastructures and can scale their resource provisions when the local cloud resources become insufficient.

1. Introduction

In this study, cloud computing refers to the dynamic and scalable provision of virtual machines (VMs) and storage services over a network, accessible through a set of well-defined interfaces. This provisioning model is often called infrastructure as a service (IaaS), an acronym used to stress how computing resources are offered as a service and paid for on a per-use basis.

This type of cloud computing represents an ideal opportunity to improve the method by which higher educational institutions (HEIs) provide computing resources to their staff, with the potential to revolutionize how administration, teaching and research activities are supported. Unfortunately, with the benefits of an increased ability to scale services dynamically to meet demand, come also several disadvantages for a service that exists purely within the public domain, i.e. outside the institutional space in a so-called public cloud. These disadvantages include, among others, compliance with legal obligations for data, inherent latency for off-site services and the general perception of reduced security. One way to overcome these limitations is for the institutions to deploy their own cloud infrastructures—i.e. private clouds. While a private cloud requires investments in hardware and implies administrative overheads, it still affords members of the institution a service-based access model to computational resources, development frameworks and applications.

The proliferation of public and private clouds creates new opportunities and challenges. For example, users can seek the best deal among freely competing public providers while still being able to access seamlessly their institutional resources through cloud interfaces. Moreover, institutions have the opportunity to maximize the usage of their infrastructure, thanks to virtualization and the rationalization of the access and management interfaces offered by cloud computing. Nonetheless, challenges emerge when users belong to multiple institutions or projects. In this case, the resources of private and public providers need to be integrated to create some sort of cloud federation. This is a demanding technological and administrative task, especially when considering that very few standards are currently available or implemented for cloud computing.

Our project called 'flexible services for the support of research' (FleSSR) addressed some of these challenges by deploying a prototype of loosely coupled private and public clouds. This prototype is then used to support both on-demand storage facilities spanning multiple academic institutions or cloud domains, and academic research requiring remote, multi-platform software development farms.

The FleSSR project marks a departure from a concept of resource sharing based on a tight interoperability among cloud fabrics [1,2]. Instead of arguing for a direct collaboration between the cloud management platforms, FleSSR endorses a brokered approach. The type of federation proposed is similar to those described, for example, in recent studies [3–5], but imposes virtually no requirements on the cloud management layer. As a consequence, the capabilities usually associated to a federation of clouds—e.g. resource management and balancing, service level agreement matching, distributed user authentication and authorization—are implemented via a brokering layer and not by means of direct integration of multiple cloud zones or modifications to the exposed cloud interfaces.

On-demand storage resources and multi-platform software development farms are ideal candidates for cloud resources [6,7], and FleSSR addresses both use cases by relying only on vanilla installations of open-source cloud products. Furthermore, compared with other solutions, the design proposed for the provisioning of on-demand storage resources allows for the deployment of multiple interfaces that the user can easily tailor or expand.

The paper is divided into seven sections. Following the introduction, the model of cloud federation is discussed in §2, and then the cloud infrastructures deployed for FleSSR are reviewed in §3. Details are offered about the project requirements and how the chosen architecture satisfies them. Special attention is given to the Eucalyptus cloud framework, to the eZeel broker and to the technologies adopted for monitoring, accounting and user authentication and authorization. Section 4 is dedicated to the two use cases addressed by FleSSR: the development of a flexible

storage service specifically tailored for the HEI users, and a multi-platform software development service that addresses the need for a highly scalable platform for the automatic testing of software on multiple operating systems and development environments. Sections 5 and 6 expound on the design and implementation details of the flexible storage service and the multi-platform software development service. Their deployment scenarios are analysed focusing on the experience of the staff at the Environmental Systems Science Centre (ESSC) and the Square Kilometre Array (SKA) Oxford research group, two user communities that tested both services. Finally, §7 summarizes the project's achievements outlining some promising future developments.

2. Cloud federation

A prototype of a loosely coupled federation of clouds is required to enable the creation of a coherent system as viewed by a user within the FleSSR project. To counter possible problems of trust and security with external providers, while allowing for local management of potentially large quantities of data, the system has to integrate both private institutional clouds and scalable public clouds. This infrastructure configuration, known also as a hybrid cloud, would provide HEI users with the flexibility to run niche or critical services locally while migrating other services remotely as is required by research, teaching and administrative activities.

As seen in §1, several models have been suggested for a federation of clouds, depending on the available resources, the requirements to be satisfied and the deployed technologies. One such solution is the creation of a brokering layer through which users access, manage and use cloud resources. Such a broker needs to be able to interface with a heterogeneous set of IaaS implementations, as it would be unrealistically restrictive to assume that each provider, especially in the HEI context, would have to deploy a specific flavour of IaaS. The broker should be able also to cache the authorization and authentication (AA) tokens so that the users would be able to use a single AA model/token to access the cloud resources on all the federated infrastructures. Within the FleSSR project, the broker called 'eZeel' developed by EoverI [8] satisfied all these requirements with the added benefit of offering an application programming interface (API) to the developers of the project use cases.

A hybrid-federated scenario requires also accounting for resource utilization among several independent institutions and user communities. The accounting system should be able to take information from multiple cloud platforms and, therefore, should support appropriate standards. A standard-based accounting system would also be useful for the future integration of grid and cloud resources at a national and international level. For example, the account records collected within the hybrid cloud could be submitted to the accounting services of national e-infrastructures like the National Grid Service (NGS) user accounting system [9]. An analogous approach should also be endorsed for monitoring the hybrid cloud.

3. Cloud infrastructures

Several manufacturers and open-source projects offer software stacks to deploy a 'private cloud'. Common examples are VMware vCloud, OpenNebula, Nimbus, Openstack and Eucalyptus [10–14]. Within FleSSR, the choice was constrained by the following requirements:

1. each cloud infrastructure had to be built with open-source software, thereby allowing developers within FleSSR to integrate and extend the software stack;
2. a management interface compatible with the Amazon web services (AWS) API [15] had to be offered so that tooling originally designed for AWS could be used;
3. the project had to offer precise indications on how to deploy production-grade cloud infrastructures to be supported in the long term;
4. the cloud infrastructures had to be used by a number of user communities both for specific use cases and as vanilla IaaS;

5. the cloud infrastructures had to be accessible through some form of standardized and federated AA mechanism; and
6. the use of each cloud resource had to be recorded and accounted for.

The Eucalyptus software stack packaged by Canonical—a FleSSR project partner—within Ubuntu [16] was chosen to provide the FleSSR cloud infrastructures as it satisfied requirements 1, 2 and 3. Furthermore, Eucalyptus can be integrated with a brokering system so as to satisfy requirements 4 and 5, and make available enough logging information to develop an accounting solution to satisfy requirement 6.

Eucalyptus is a modular cloud solution designed around five main components called 'controllers' [17] that communicate via the Simple Object Access Protocol with web service-security. The cloud controller (CLC) offers Amazon's elastic cloud computing—compatible interfaces and a web portal useful mainly to manage users and their credentials. The Walrus Controller (Walrus) implements an Amazon's simple storage service—compatible interface while the cluster controller (CC) provides scheduling among clusters and networking for the VMs. The storage controller (SC) provides elastic block storage (EBS)—style storage and, finally, the node controller (NC) drives the chosen hypervisor (KVM or Xen [18,19]) on each node where the VMs are run.

Eucalyptus offers a set of functionalities that are loosely comparable to some of those implemented by AWS. Such functionalities include the following:

— security groups: users can define groups, with access rules indicating what port can be accessed from which source Internet Protocol (IP). Multiple VMs can then be instantiated and associated with a defined group;
— elastic IPs: one or more public IP(s) can be associated with a running VM by the CC; and
— VM isolation: VM network traffic is isolated from the traffic of VMs belonging to a different security group.

Eucalyptus can be installed on several hardware configurations where a number of physical machines are available or a degree of controller separation is desirable. Furthermore, community documentation is available [20,21] for both the minimum hardware requirements and the installation process.

Through the project, three Eucalyptus infrastructures have been deployed: one at Eduserv, one at the University of Reading and one at the University of Oxford. The development required by the two use cases has been conducted on the Reading and Oxford infrastructures. The Eduserv installation has instead been used as a pre-production public cloud. Together, these three cloud deployments have been working as a hybrid infrastructure.

(a) Brokering layer

The application code for each use case communicates with the underlying clouds using the eZeel API. This allows for resource composition to be changed at any point without any required intervention on the internal application code.

The design of the broker relies on the clear separation of the application from the resources on which it may run. This introduces the concepts of application cloud and resource cloud. An application cloud is the overall application, made up of the simple atomic service components. These atomic components can be run on separate cloud instances. A resource cloud is a collection of resource providers that should remain agnostic to the underlying provider, thereby allowing the application to use resources from whichever provider meets the needs of the application at any given time. Which provider or set of providers can be used depends on the constraints of the application (for instance, staying within some geographical or legal region) and also on the financial arrangements defined by the application owner. Once this financial relationship with the provider is established, the eZeel layer presents the user with a menu of available resources.

(b) Resource selection

Once a pool of resources is defined, an application can select and allocate suitable resources within it. Owing to its provider-neutral design, eZeel takes a discovery-based approach where an application attempts to find resources that meet its requirements. This model of resource description and selection is similar to the Condor ClassAd system [22], where requirements are expressed as constraints on acceptable resources and resource configurations in order to satisfy the needs of the application. Each constraint is an attribute–value pair, for example, that a particular central processing unit (CPU) type, random access memory (RAM) size, operating system version or storage access type is to be used. Through aggregation of these individual values, complex requirement relationships can be built. An illustration of this is given in figure 1.

The capability to specify constraints to select a resource rather than mandating a design-time-specified solution is very powerful. It allows the developer to use the best available resource solution at the application's runtime, on the basis of their current situational requirements. Thus, new capabilities added by a provider can be incorporated without changing the application code.

Once a collection of suitable resources ('offers') has been identified as meeting the requirements of the application, they can be filtered by a cost model to select the best match for the application. Within the FleSSR project, simple cost models were used for the use cases, leaving the definition of more fine-grained cost policies for a future, production phase.

(c) Managed providers and simple application programming interfaces

Within the eZeel system, the user application does not interface directly with a cloud provider but instead with the eZeel manager provider. This allows the definition of a simple provider agnostic eZeel API [8]. This approach has the following benefits:

- clients can be lightweight because they do not need all possible cloud provider libraries and communication uses the higher-level eZeel API;
- clients do not need to be updated as the back-end cloud provider APIs, resources and costs change;
- managed providers can enforce policies and rules at a single cloud or multi-cloud layer, which is not possible with any cloud providers or other multi-provider abstraction layers; and
- credentials for individual end resource providers may be shared without divulging them to users (especially useful when combined with policies and rules).

eZeel managed providers existed for AWS, Rightscale, Flexiscale and VMWare. The EoverI team worked closely with the other project members in order to develop a provider for Eucalyptus so that the adoption and deployment of eZeel would not require any modification to the three Eucalyptus infrastructures of the project.

(d) User and service credentials

eZeel uses strongly typed objects for its credential system. Credentials are different for direct and managed providers:

- managed providers take managed provider credential types (such as a user certificate from the NGS); and
- direct providers take custom credential types (for instance, AWSCredential, EucalyptusCredential, FlexiscaleCredential).

These different types of credential are located within the eZeel service in the credential store. In order to store credentials separately from code, eZeel provides a number of credential store types (as well as an interface allowing others to implement their own credential store).

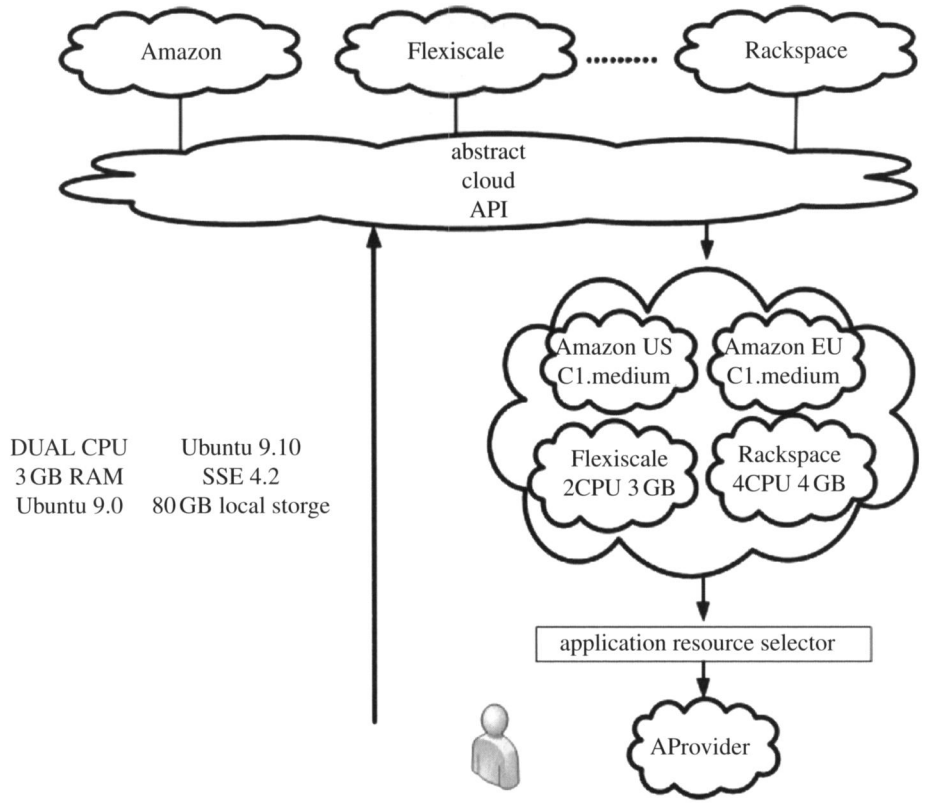

Figure 1. Composite user application requirements are passed into eZeel, which then interrogates each resource provider and returns a set of offers for available instances. (Online version in colour.)

Managed providers must have some way of mapping the incoming user onto a credential for the Provider they manage. This mapping mechanism is completely customizable, though the most common mechanisms are:

— all-to-one: all permitted users are mapped onto a single credential;
— direct: each permitted user is mapped onto their own credential; and
— complex: permitted users are mapped onto credentials, each credential having zero or more permitted users mapped onto it.

This facility allows for the use of remote cloud resources that would not normally support authorization mechanisms around groups and limited sharing such as through virtual organizations.

(e) Management, monitor and accounting

Aside from interacting with cloud resources through an application API, we must also be able to manage remote instances directly on the cloud. The management of the instances and overall Eucalyptus cloud management are separate issues. In order to manage and monitor Eucalyptus, we use a Firefox add-on called Hybridfox [23], shell scripts to filter and monitor relevant log files, the Eucatools commands, Ganglia [24] and Nagios with tailored probes [25].

Accounting and per-user resource quota allocation are provided only with the Enterprise version of Eucalyptus. As a consequence, an accounting system has been developed within FleSSR.

The accounting solution for Eucalyptus is designed to satisfy three main requirements:

— the source code of Eucalyptus should not be modified;
— the accounting records should be consistent with the Open Grid Forum (OGF) usage record recommendation [26]; and
— the accounting records should be updated to the current NGS facilities, operating an OGF resource usage service compliant accounting repository [27].

The accounting system is implemented in Perl and consists of three modules and one database:

— Aggregator module: aggregates the Eucalyptus log files by filtering the available log entries and storing only the relevant data;
— Parser module: parses the aggregated log files in order to populate the usage record database (URDb);
— URDb: stores all the users and usage data. Each registered Eucalyptus user is associated with his VM, EBS and Walrus usage; and
— Interface module: polls the URDb compiling the usage records. The format of the usage record is defined in this module following what is required by the adopted standard.

4. Use cases: flexible storage service and multi-platform software development service

Within the research community, there is a significant pressure to use computational infrastructure as efficiently as possible. This includes minimizing purchases of dedicated hardware resources and their associated support costs, and maximizing the utilization of the existing resources. The FleSSR use cases are two paradigmatic examples of situations in which maximization and reusability of resources are critical: data-intensive research and multi-platform development. For each of these, the amount of used resources is difficult to predict, quantify, manage and maintain. The availability of cloud resources provides a simple mechanism by which the user gains control over when and how many resources are deployed. The user avoids unnecessary interactions with information technology (IT) support staff, and hardware-related investments are minimized.

Typically, in data-intensive research projects, raw data need to be readily available to the researchers as they are collected. Once the raw data have been analysed and annotated, they need to be archived for release through some form of institutional- or subject-specific repository. This workflow promotes the fragmentation of data resources as every single project tends to purchase a significant number of low-specification resources for immediate data processing. Such resources cannot be efficiently shared across multiple research projects, and are not suitable for long-term storage and the dissemination of value-added data.

Analogous problems exist for large research projects involving multi-platform software development. Such projects tend to require a large number of operating system (OS) images that are constantly updated, synchronized and available to the whole development team. These requirements impose costly administrative overheads to every research team and are seldom satisfied by the typical departmental IT resources. These issues lead to inefficient fragmentation and expensive duplication of the computational resources across research groups, as every development team has to have such resources locally available.

Within FleSSR, the development of each case study required a specific tailoring and configuration of the eZeel brokering interface. In this way, the inner complexities of users interacting with multiple cloud providers could be hidden. At the application level, users are provided with a single, consistent interface that allows for x.509-based authentication and authorization, cloud resources management, policy specification and testing of the cloud infrastructure performance.

The full system developed and deployed for FleSSR is shown in figure 2.

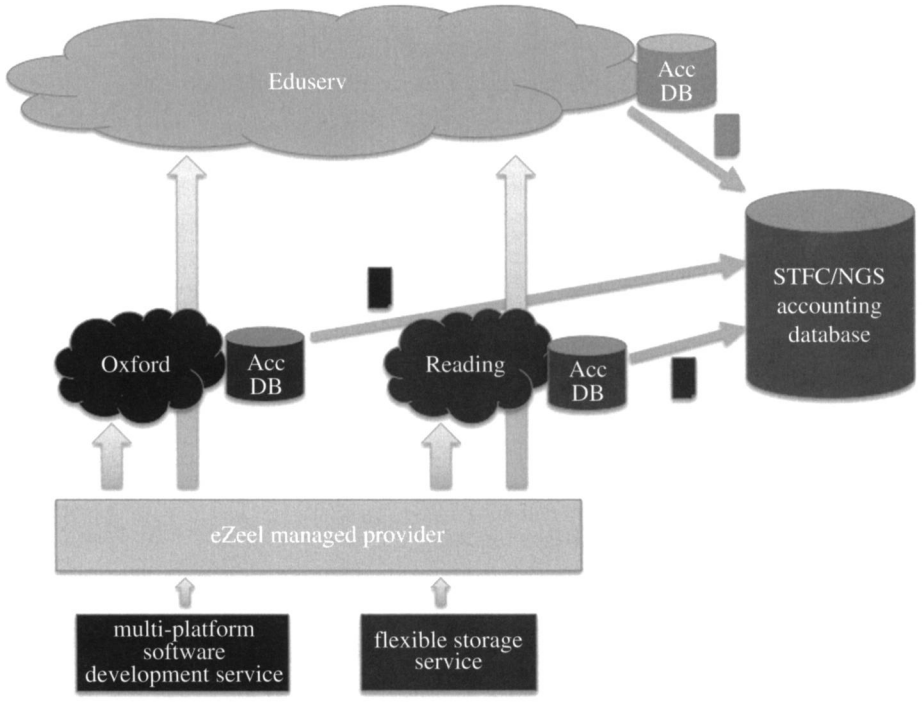

Figure 2. Overall system architecture showing instances and storage blocks being instantiated from the user interface, interacting through eZeel with the hybrid cloud. The accounting information from these systems is then centrally collected through the National Grid Service accounting portal. AccDB, Accounting Database; STFC, Science & Technology Facilities Council. (Online version in colour.)

5. Flexible storage service

The flexile storage service consists of three components: the data management controller (DMC); a tailored OS image from which to instantiate VMs; and a set of EBS-like volumes to be attached to the VMs (figure 3). The DMC is a user-facing front-end for the eZeel broker. Written in Java, the DMC is loaded into a browser as an applet, but it could be easily deployed as a stand-alone application, depending on what requirements need to be satisfied.

The DMC allows a user to create a storage volume specifying its size, the period of time for which it should be available and one or more interfaces/protocols to access it—e.g. WebDAV, file transfer protocol (FTP), secure copy or rsync. Once this initial step is completed, the user is presented with the list of his active or inactive volumes and is given the choice of whether to activate or deactivate them. A volume is active when it is accessible through the specified interfaces, inactive otherwise.

From the cloud infrastructure standpoint, a volume is active when it is attached to a VM and it is exposed through a public IP. The VM is instantiated from a customized Linux-based image configured to automatically mount a given volume and start the services needed to expose it. The volume identification and the name of the services that need to be started are pulled out by the instances from the cloud infrastructure information system. In this way, just a single, customized OS image is needed, no matter what volumes have to be mounted or set of services to be started. Analogously, a single VM can expose multiple volumes belonging to the same user.

Authentication and authorization for the DMC are managed through eZeel and based on x.509 digital certificates. For the FleSSR prototype, the UK e-Science Certificate Authority (CA) is recognized but adding multiple CAs requires no further development. While FleSSR guarantees access to the cloud management interfaces and to the DMC with a single user certificate, further

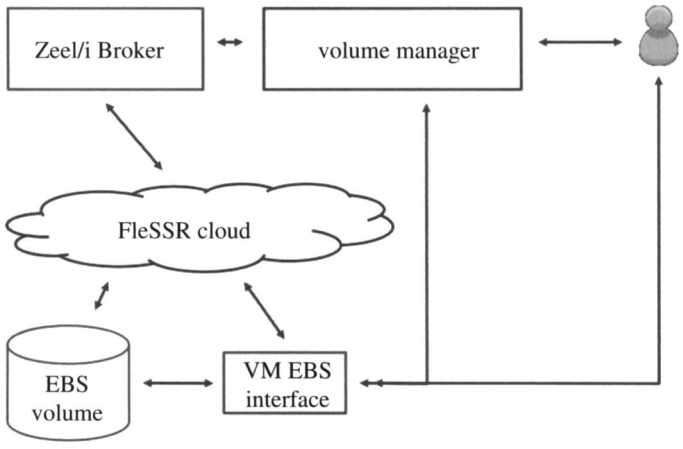

Figure 3. Operation of the flexible storage service. (Online version in colour.)

work would be needed to use that same certificate to access the instantiated VMs or the data exposed through them. As a consequence, users have to manage at least three authentication and authorization tokens, a common shortcoming of currently available cloud infrastructures.

The VM to which the volume is attached is instantiated with the privileges and the public key (if any) of the volume owner. In this way, the user retains full control over the access management of his volumes and, if required, is able to securely log into the running VM via Secure Shell. Exposing the volume through a VM allows for maximum flexibility, not only for the type of interfaces/services used to expose the user's data, but also for further customizing the data management process.

This flexibility is particularly important when dealing with HEI users because they present a wide range of requirements and technical skills. The requirement elicitation process conducted within FleSSR focused specifically on academic researchers and showed that, while some users simply need a cheap, efficient and simple-to-use replacement for their external universal serial bus hard drives, others have far more demanding requirements. Users with advanced technical skills or complex use cases need to customize a VM to expose their data through, for example, web services, distributed file systems or, in other cases, need to automate the pre-processing of their data before exposing them.

The whole design of the flexible storage service has been devised to maximize the opportunity for customization. Being a front-end to a brokering library, the DMC does not implement any low-level functionality required to access the cloud infrastructure, to manage a VM, an EBS volume or a public IP address, or to define the access rules to the data services. For this reason, the DMC can be easily extended and further tailored for the specific requirements of each research project.

(a) Architecture

The development architecture for the flexible storage service is based on Ubuntu 11.04 Cloud Server running Eucalyptus 2.01, installed on three physical machines (figure 4):

— one Dell 2950 as combined CC, CLC, SC and Walrus; and
— two Dell R610 as NCs and shared storage.

Gigabit Ethernet is used among the servers and for external connections.

Overall, the Reading Cloud Infrastructure supports up to 48 VM instances and a total of 3 TB of storage with 1 TB of serial attached small computer system interface storage per node shared via the network file system (NFS).

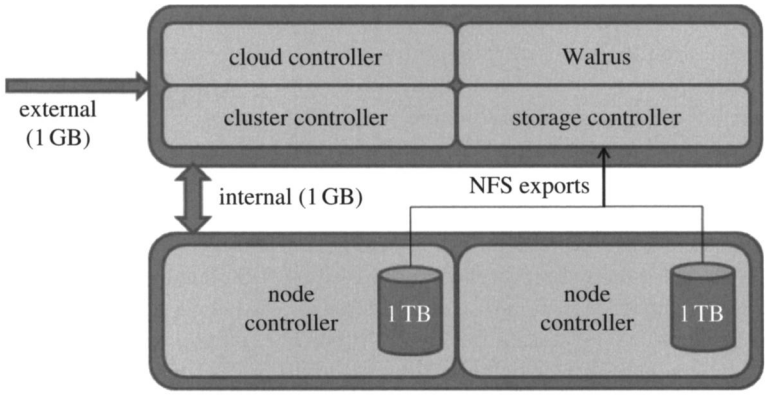

Figure 4. Architecture for the development of the flexible storage service use case. (Online version in colour.)

The storage infrastructure is kept within the servers, but in a production architecture it would be served through a dedicated, fast connection—e.g. storage area network (SAN) or 10 GB Ethernet—to the SC. For a prototype, the two NCs have a relatively large amount of local disk available and are well suited to provide the capacity and throughput required by the FleSSR use cases.

As usual in a Eucalyptus deployment, the SC service controls two repositories, one for 'buckets' and another for 'volumes'. Buckets are areas for users to store VM images. Volumes are the containers for user-defined disk volumes, also known as EBS volumes. These storage pools are considered local to the SC, with VM images being copied to and cached on the NCs when a new VM is requested. EBS volumes are always kept on the SC, instead of being dynamically shared via an Internet small computer system interface or ATA over Ethernet (ATAoE) to the NC that requires access to that particular volume.

Storage is shared via NFS and kept to a simple design of one repository per node. Owing to always being accessed centrally, the largest overhead is on the EBS storage. This would be the primary candidate for being located on a SAN in a production environment.

(b) Scenarios

The flexible storage service main use case is replacing external hard drives. This is particularly important in those research environments in which data need to be retained after their initial curators have moved on and are no longer available. With the increased demand from funding bodies to retain and make available data produced by funded projects, academic users and research groups need viable solutions for digital data curation and maintainability. Among the technologies required to address such requests—i.e. metadata, annotation systems, dedicated catalogues, standardized archiving, access procedures and protocols—the storage service developed within FleSSR offers a scalable, distributed and on-demand storage facility where data can be stored, collected and possibly archived. The long-term goal is to use a production version of this storage facility as the base for the development of the aforementioned technologies for data curation and maintainability.

One of the main scenarios addressed by the flexible storage service is one in which datasets that are large enough not to be feasibly replicated across cloud zones, need to be accessed from multiple geographical locations or by users distributed across research institutions. This scenario applies to datasets exceeding a few terabytes that would be moved through common network infrastructures with a speed in the range of no more than a gigabit per second. The flexible storage service allows for remote access to these datasets without necessarily requiring their replication across multiple cloud zones. The scenario sees a user creating terabytes of data by

means of the cloud infrastructure or by physically moving them. Data are stored in one or more EBS volumes that are then exposed through the flexible storage service as described above and with user-definable access rules. It should be noted that it would still be feasible to replicate the data when not in use through an automated replication process. Analogously, once the stated lifetime of the storage is reached, the EBS volume(s) could be moved, along with the VM image, to a public cloud, thereby saving local storage for high duty cycle instances while maintaining data accessible.

Another scenario addressed by the flexible storage service is the need for user communities to adopt GlusterFS [28] for their cloud-based storage resources. Staff at the ESSC at the University of Reading tested the Gluster-distributed storage system on the FleSSR cloud infrastructures. Many researchers at the ESSC have datasets that are hundreds of gigabytes in size, and the total amount of data stored on the ESSC's servers currently exceeds 100 TB. This capacity has been built up over several years by the acquisition of computer hardware as and when research funds were available. One result of this piecemeal expansion is that large datasets frequently span two or more separate servers. For this reason, the ESSC has implemented a distributed storage system called GlusterFS to manage the data effectively.

In the ESSC's GlusterFS storage cluster, the servers are deployed as replicated pairs, and each pair is divided up into a number of storage units called 'bricks'. A volume typically consists of two or more replicated bricks, with files evenly distributed between the bricks on several servers. Each volume is exported via NFS and also via the GlusterFS native client, which is best used for applications involving a high level of concurrent data access. The replication feature allows all the files in each volume to remain accessible in the event of the failure or scheduled maintenance of one server in each pair. Most of the data is the output of computer models running on local or remote servers and clusters, but there is also a significant amount of observational data and products of data analysis and processing utilities.

In order to make effective use of FleSSR's storage facilities, the ESSC would need to be able to aggregate the capacity of several EBS volumes and to have the flexibility to increase their size according to demand. An increasing number of GlusterFS users are using commercial cloud storage infrastructures such as AWS, and recent versions of the software have enhanced features specifically for this type of usage. Therefore, it was sensible for GlusterFS to be involved in the ESSC's FleSSR storage tests. A long-term aspiration is to extend the ESSC's existing storage cluster into the FleSSR cloud to cater for sudden or short-term increases in demand for storage. To test the effectiveness of FleSSR for storing ESSC type research data, GlusterFS volumes have been set up on the Reading and Eduserv FleSSR clouds. The main concerns are performance and reliability, especially given the physical distance between the ESSC and Eduserv's data centre in Swindon.

Deploying GlusterFS is relatively straightforward, assisted by the availability of Debian class installation packages. The GlusterFS instances are deployed by being attached to a single 100 GB EBS block. This is the largest EBS block size available on the Reading and Eduserv clouds during the project, but there is no reason why this block size could not be increased in the future to facilitate the management of volumes several TB in size.

Two 500 GB volumes have been created, one at Reading and one at Eduserv, each consisting of five EBS volumes. GlusterFS replication is not tested because all the EBS volumes in each GlusterFS volume are physically located on the same server. Although it is beyond the scope of the FleSSR project, there are several potential uses of replication between two or more clouds or cloud providers, a technique that is sometimes known as geo-replication. This could, for example, be used to safeguard against data loss or temporary unavailability involving one cloud, or to allow two or more groups of users a large distance apart to share the same data.

The early tests of the Reading and Eduserv storage volumes are positive. The Eucalyptus cloud security features are flexible enough to allow both volumes to be mounted in the ESSC's hierarchical network file system, enabling users to access the FleSSR data in the same way as data stored on the ESSC's local storage cluster and conventional file servers. The Reading FleSSR volume is indistinguishable from the local storage volumes in terms of performance, with write speeds of approximately $30\,\mathrm{MB\,s^{-1}}$ measured on average for a range of file sizes up to several

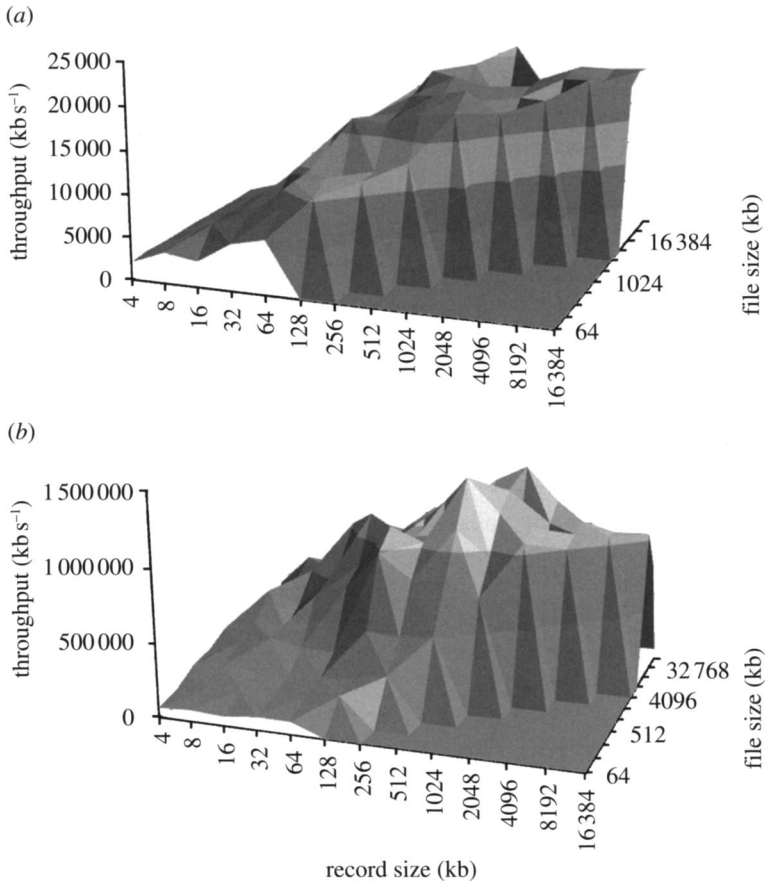

Figure 5. Performance of GlusterFS across multiple cloud instances in FleSSR: (*a*) random write and (*b*) random read. (Online version in colour.)

gigabytes per file. Access to the Eduserv volume is much slower, as expected, with typical write speeds of around 5 MB s^{-1}. This level of performance would not rule out direct access to the data by computer models and data analysis utilities, but it is more likely that such volumes would be more useful as archive repositories of infrequently used data (figure 5).

Unfortunately, the unreliability of Eucalyptus prevents a more thorough evaluation of the effectiveness of the FleSSR storage facilities by ESSC researchers. Despite several debugging attempts by system administrators, followed by repeated GlusterFS volume deployments (aided by shell scripts designed to automate parts of the process), the system cannot withstand sustained data transfer for more than a few hours at a time, and the 500 GB volumes have not yet come close to being filled to capacity. However, the basic data transfer tests carried out suggest that FleSSR cloud storage could be used effectively by researchers at the ESSC to supplement, or even replace, the local storage cluster, especially when considering that Eucalyptus could easily be replaced with more stable alternatives that are becoming now available—e.g. OpenStack or OpenNebula.

6. Multi-platform software development service

Within the research community, there are a significant number of infrastructure projects with sizable software and computing budgets. One of the reasons for such a relevant budget is that a large amount of new software development and software support is undertaken upon which

many communities of end users depend. Until recently, large international projects maintained control over the software development cycle by limiting the number of platforms onto which their software could be deployed in a supported manner. This is a pragmatic choice, though it has led, on a number of occasions, to a suboptimal situation where users and resource providers were forced to rely on outdated and nearly unsupported operating systems. More recently, these projects have recognized that such an approach is not sustainable in the long term, but are struggling to find the financial capacity needed to provide support for all the platforms that their researchers may need to use. As a consequence, a technological solution has become increasingly desirable to simplify and reduce the costs of developing and maintaining multi-platform software within these communities.

A prime example of this trend is the now formally established European Strategic Framework for Research Infrastructures project named the SKA [29]. This project has a global participation with many loosely connected researchers and several groups involved in building the arrays and developing the required software systems. In setting up the SKA, it has been decided that no limits should be imposed on what platform the software developed within the project will run. Therefore, this software will have to be tested on multiple platforms during its development process. The testing procedure will involve both unit and integration tests to be performed on a nightly basis and on a fresh OS environment.

The SKA requirements for multi-platform software testing are particularly well suited to virtualization, cloud computing and, in particular, to the FleSSR hybrid cloud infrastructure. The SKA development effort is highly distributed across several largely independent groups with limited local resources. In this condition, the risk of effort duplication and inefficient use of local resources is very high. While virtualization clearly serves the purpose to run efficiently multiple OS platforms, cloud computing and the type of brokering among cloud providers proposed by FleSSR offer the opportunity to each development group to retain a degree of control over the testing procedures of their software while benefitting from a readily available scalability. This scalability is obtained thanks to the brokering among local—i.e. private—cloud infrastructures managed by different groups belonging to the SKA project and, if needed, by relying on public clouds such as Eduserv, AWS or other providers for which an eZeel managed provider can be developed.

In order to leverage the benefits of a hybrid cloud for the SKA, a multi-platform software development service has been developed within the FleSSR project. The architecture used to develop the use case is very similar to the one deployed in Reading. A diagram of the Eucalyptus installation as used in Oxford is available in figure 6.

(a) Scenarios

When supplying a software tool to an end user, the actual coding is only half the battle. Installing a working executable on the user's machine is also a significant challenge, particularly when there are a large number of dependencies. As this can be a non-trivial task, it is important to test this installation process on a machine that resembles that of the end user, which is very unlikely to have all the products—or the same version of the products—that a developer might have installed on his workstation. A system for the automated provision of an environment resembling that of an end user would then be a very useful facility that could seriously reduce application support costs and, ultimately, the end user's frustration.

Developing software for multiple platforms is often a requirement. The usual reasons for this include: allowing the end user to work on their favourite platform, complying with the requirements of their employers IT environment and reducing the risks of vendor lock-in. In addition, multi-platform development can significantly increase the quality of code as it requires better design.

Developers need access to all the supported architectures in order to test and debug their code. It is unreasonable for them to be forced to maintain multiple platforms, as there could be over 20 of these just when considering only popular Linux distributions, though each project still will

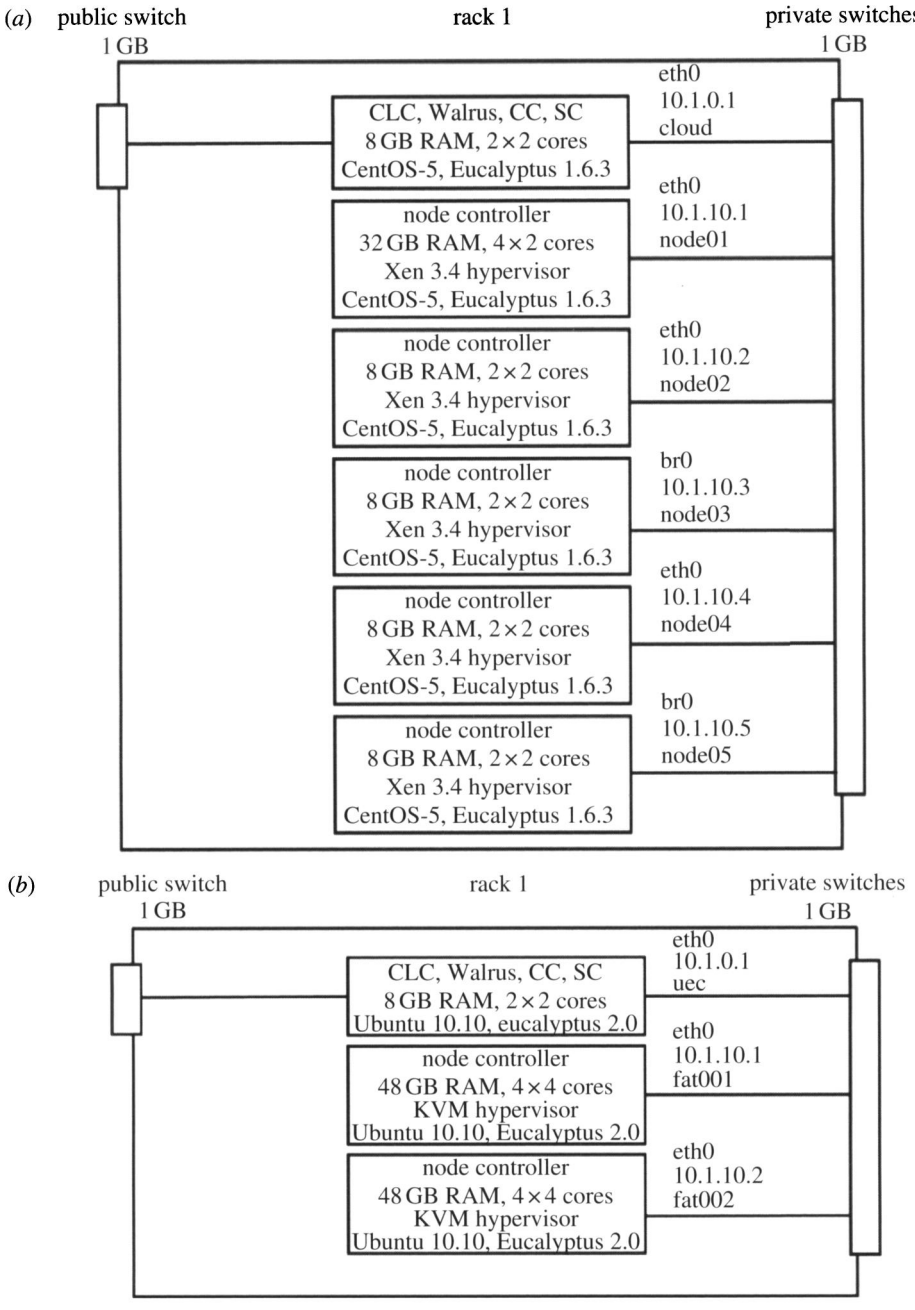

Figure 6. (*a*) Eucalyptus infrastructure based on CentOS-5 and (*b*) Eucalyptus 2.0.2 infrastructure based on Ubuntu 10.10.

invest significantly in hardware for each of their developers. This is not just a hardware issue, but also a system administration issue, as each platform has to be kept up to date and installed with the software needed for the development. A central service to provide these platforms on demand is clearly a great time and resource saver, allowing developers to concentrate on writing and testing their code.

As well as product releases, the code base is often rebuilt and tested every night to ensure that no bugs have slipped in unnoticed. A service to build and test on all the required platforms is therefore needed.

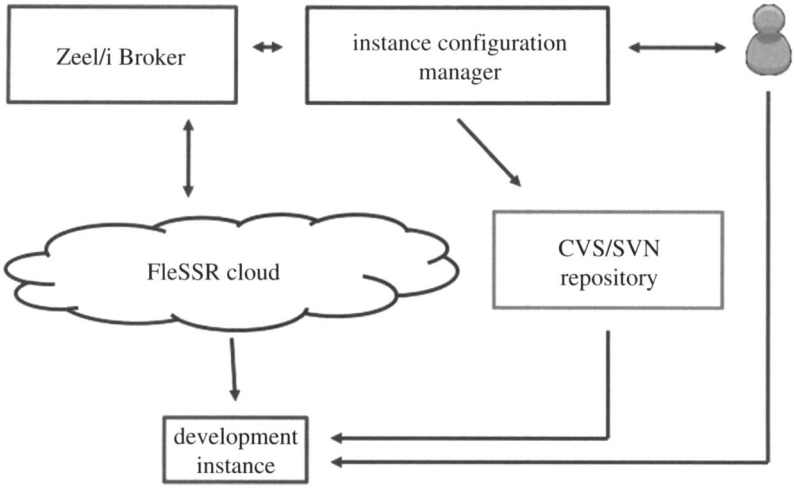

Figure 7. Provision of a single instance for code support and bug investigation. CVS, concurrent versions system; SVN, apache subversion. (Online version in colour.)

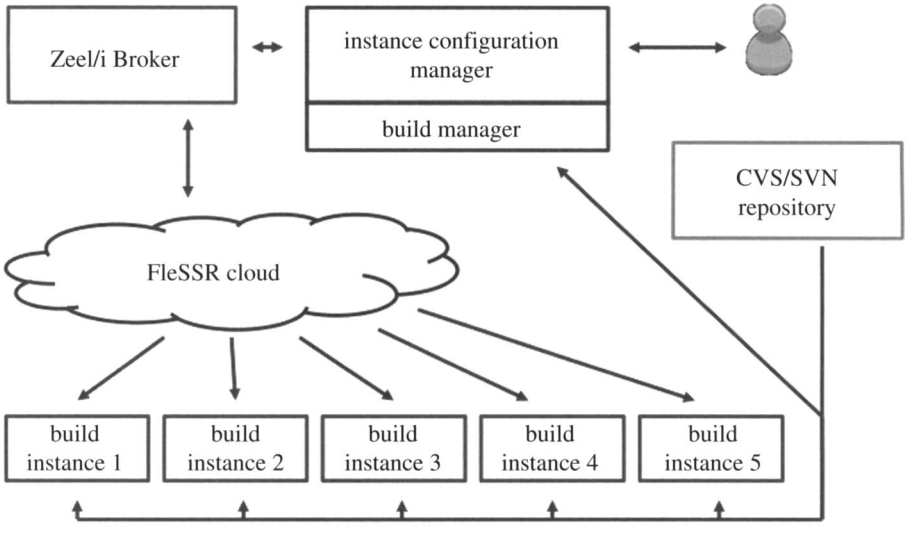

Figure 8. Description of software build operation with compilation instances running on public cloud. CVS, concurrent versions system; SVN, apache subversion. (Online version in colour.)

(b) Design

There are two different stages to the use case as designed. The first is for the easy provision of an exemplar instance into the cloud with the ability to choose the type of operating system that will be installed onto the system along with dependent software and libraries for the application that is to be tested (figure 7).

The second is the provision under which the application developer is able to launch—using some form of automated framework such as Buildbot [30] or similar—a number of predefined compilation instances using a set of different platforms. After compilation has been performed, the application can have unit and integration tests run against it to test functionality (figure 8).

A multi-platform software development service should implement the following operational steps in order to simplify the interactions between the software developer and the application environment.

The project workflow as first designed is:

— select project,

(i) select dependant codes,
(ii) select build platforms, OS and architecture, and
(iii) select project source code,

— start instances,

(i) based on select platform list start instances,
(ii) on running instances download and install dependencies and source code,
(iii) start build process, and
(iv) stream standard out and standard error from each build instance, and

— compile and report outputs,

(i) pack complete products,
(ii) report errors and ship complete builds, and
(iii) allow access to error plagued builds directly on instances.

(c) The multi-platform publishing tool

Multi-platform publishing (MPP) is a tool developed within the radio astronomy community to ease the complexity and cost of deploying computer software across multiple platforms.

Each platform has its own conventions, standards, package managers and packaging formats for deploying software. MPP allows the user to describe a software product in generic terms (e.g. this is a binary, this is a library, this is documentation, it requires these dependencies), and will produce a suitable package tailored to each supported platform, which can be easily installed by the user in the normal way native to that platform.

It consists of three process layers: Build, Test and Publish. MPP is designed to support the entire release and testing processes. Each of these processes can be launched with a single MPP command.

(d) Build

With a description file, the source code and access to the supported platforms, MPP will create suitable packages containing the build products. These packages are transferred from the platform on which they were built to a central repository for testing and deployment. Table 1 shows two different descriptions files, one for a C compilation and another for a python build.

(e) Test

For each supported platform, it is important to try out the packages built in the previous stage to ensure they work for a typical user. Clean images for each platform are used (i.e. one without any dependencies or development products installed), and any problems with the installation are reported back for fixing. Additional post-installation testing scripts can also be specified.

(f) Publish

Now that the packages have been tested, they are ready to be distributed to the wider community. MPP allows you to publish these packages to suitable repositories (again platform specific), from

Table 1. Two examples of build configuration file, one a C-based application (left-hand side) and the other a python application (right-hand side), both for the SKA project.

```
[project]
name=oskar-simulator
licence=BSD
description=The Oskar SKA Station
Beamforming Simulator Backend

[platforms]
ubuntu_8_10-64bit
ubuntu_8_10-32bit
ubuntu_8_04-64bit
ubuntu_8_04-32bit
ubuntu_9_04-64bit
ubuntu_9_04-32bit
openSuse_11_1-32bit
openSuse_11_1-64bit
fedora_11-64bit
fedora_11-32bit
centos_5_3-32bit
centos_5_3-64bit

[install]
/*

[dependencies::build]
cmake
c
c++

[dependencies]
cblas
openmpi
xml2

[dependencies::runtime]
server-ssh

[build]
cmd=cmake                                -
DCMAKE_INSTALL_PREFIX=${prefix}/usr   .   &&
make && make install

[build::ubuntu_8_04]
useRepository=oxford_apt:pre-release

[build::openSuse_11_1]
useRepository=oxford_meqtrees_yum:pre-
release
cmd=cmake                                -
DMPI_COMPILER=${install::lib}/mpi/gcc/openmp
i/bin/mpic++                             -
DCMAKE_INSTALL_PREFIX=${prefix}/usr   .   &&
make && make install

[build::openSuse_11_1-64bit]
cmd=cmake                                -
DMPI_COMPILER=${install::lib}/mpi/gcc/openmp
i/bin/mpic++                             -
DCMAKE_INCLUDE_PATH=${install::lib}/mpi/gcc/
include                                  -
DCMAKE_LIBRARY_PATH=${install::lib}/mpi/gcc/
openmpi/lib64                            -
DCMAKE_INSTALL_PREFIX=${prefix}/usr   .   &&
make && make install

[build::centos_5_3]
useRepository=oxford_meqtrees_yum:pre-
release
cmd=cmake                                -
DCMAKE_C_COMPILER=${pack::openmpi::compiler_
cc}                                      -
DMPI_INCLUDE_PATH=${pack::openmpi::include}
-DMPI_LIBRARY=${pack::openmpi::lib}       -
DCMAKE_INSTALL_PREFIX=${prefix}/usr   .   &&
make && make install

[code]
srcDirectory=1.1.1
srcPack=oskar-${version}.tar.bz2
```

```
[project]
description=Log generation Tool
licence=GPL

[description]
purr will watch a directory for any new files. If
its a type it can recognise it will analyse the
file and generate appropriate images etc in a
html format log file.

[platforms]
ubuntu_9_10-64bit
ubuntu_9_10-32bit
ubuntu_9_04-64bit
ubuntu_9_04-32bit
ubuntu_8_10-64bit
ubuntu_8_10-32bit
ubuntu_8_04-64bit
ubuntu_8_04-32bit
openSuse_11_1-64bit
openSuse_11_1-32bit
fedora_11-64bit
fedora_11-32bit
centos_5_3-32bit
centos_5_3-64bit

[dependencies::runtime]
python-qt4
python-imaging
python-pyfits
python-tk

[install_link::bin]
purr=${install::python_lib}/Purr/Purr/purr.py

[install::python_lib]
purr.pth

[install::python_lib::Purr/Purr]
Purr/*.py

[install::python_lib::Purr/Kittens]
Kittens/*.py

[install::python_lib::Purr/icons/purr]
icons/purr/*.png
icons/purr/*.xpm

[install::python_lib::Purr/Purr/Plugins]
Purr/Plugins/*.py


[install::python_lib::Purr/Purr/Plugins/local_pyc
hart]
Purr/Plugins/local_pychart/*.py

[install::python_lib::Purr/Purr/Plugins/local_pyc
hart/afm]
Purr/Plugins/local_pychart/afm/*.py

[postinstall]
${command::python}          -m          compileall
${install::python_lib}/Purr

[preuninstall]
rm ${install::python_lib}/Purr/Purr/*.pyc
rm ${install::python_lib}/Purr/Kittens/*.pyc
rm ${install::python_lib}/Purr/Purr/Plugins/*.pyc
rm
${install::python_lib}/Purr/Purr/Plugins/local_py
chart/*.pyc

[build]
copyExpand=(purr.pth purr.pth)

[code]
srcDirectory=Purr
srcPack=purr-${version}.tar.bz2
```

which other users may install the packages on their machines directly. MPP supports multiple levels of publishing to allow you to tailor your release process (e.g. a repository for beta-testers, one for supported releases, etc.).

(g) Multi-platform publishing and the flexible services for the support of research project

Within FleSSR, MPP is offered to all software projects as a generic, centrally managed service.

MPP has been refactored so that users with suitable credentials can perform the build and test operations on platforms hosted on the cloud. When users invoke the build step, MPP requests to instantiate a separate VM for each platform type. Users can then use these VMs to test and package their software. Once the build/test cycle is completed, the VMs are shut down, freeing the cloud resources.

(h) Use case utilization and critique

The SKA consortium has not been using the system heavily due to some problems with Eucalyptus. There are teething problems owing to slow start up within the build manager of the Java VM. These issues should be infrastructure specific, and in a future production service, they could be addressed by designing, for example, the build manager to be long lived. The SKA developers have looked at porting the ideas that have been developed through this project onto VMware ESXi [31], which they already have access to within the hosting department. This porting may be taken further as the availability of this type of system further increases. There are limitations to this solution, with ESXi not being able to easily provision multiple instances from a single disk image.

As initially designed, the FleSSR system is set up so that a single user has access to the cloud broker and resources, which would pose limitations for a production environment with many users. Therefore, further work would be required to allow multiple users to easily share instances among themselves, in a controllable way and based on user attributes. This is currently a limitation shared by many of the cloud solutions that are 'all or nothing' in terms of instance and data sharing.

Originally, the use case description included a graphical user interface to operate the software build process, but the requirement elicitation process revealed that the target community of developers preferred to work with command line and text editing capability. A future piece of work would be to develop a plug-in for the popular and open-source Eclipse framework so that software builds could be quickly and easily deployed onto the multiple target platforms the developer needs to support.

7. Conclusion

Overall, the infrastructure developed by this project offers an efficient, secure and scalable design for the on-demand provision of computational and data resources. Even with all the inevitable limitations typical of a prototype, FleSSR shows how a similar solution could be scaled to production with appropriate extra development work, thereby greatly reducing duplication of resources, the overall amount of acquired hardware and the overheads of both research and administration staff.

In the described scenarios, it is fundamental to consider the economic trade-off between ownership and temporary rental of resources. The integration of private and public clouds allows for the maximization of such a trade-off. Services can run in different environments depending on their peculiar requirements. For example, services that require highly responsive connectivity or that deal with sensitive data would run on a private cloud. Other services that require a much larger infrastructure, but that are less critical, would run instead on a public cloud.

FleSSR offers an insight into the benefits and limitations of alternative models for federating clouds. The prototyped infrastructure shows the importance of distinguishing between a federation at the IaaS level and one at the brokering level. While federating cloud middleware by creating, for example, multiple zones is desirable when a single institution has direct control over all the available hardware, a federation based on a brokering layer is well suited for independent and potentially competitive institutions that own limited local resources.

FleSSR represents a first step towards the development of applications that seamlessly exploit the peculiarities of different types of cloud infrastructures. Applications might scale across private

and public clouds depending on finer-grained policies, involving not only the need for more resources but also their properties. For example, an application developed by a multi-institutional project could scale across resources belonging only to the project partners or an application could scale to public clouds only when dealing with anonymized data. The eZeel policy engine described in §3 permits the definition and endorsement of this kind of policies.

The FleSSR system is only a prototype of how a production-ready federated cloud for HEIs should be designed and deployed. Further development is needed in many critical areas. First and foremost, a solid cloud middleware is required. Eucalyptus has proved too unstable for anything but a very small prototype. In many cases, this instability has hindered the possibility to test further the software developed within FleSSR. Research is also needed to improve the security of federated clouds. In this context, the attestation of the VMs and storage volumes provenance and the development of an authentication and authorization infrastructure for federated clouds are particularly important.

Finally, FleSSR clearly showed the importance of devising dedicated technologies and paradigms for the development of cloud applications. Developers need to produce applications designed to run natively on multiple clouds. Only in this way will the promising benefits of federated cloud computing be fully exploited.

This project was funded through the UK JISC Flexible Services programme.

References

1. Rochwerger B *et al.* 2009 The reservoir model and architecture for open federated cloud computing. *IBM J. Res. Dev.* **53**, 535–545. (doi:10.1147/JRD.2009.5429058)
2. Buyya R, Ranjan R, Calheiros R. 2010 InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In *Proc. 10th Int. Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2010), Busan, South Korea, 21–23 May 2010*, pp. 328–336. Berlin, Germany: Springer.
3. Celesti A, Tusa F, Villari M, Puliafito A. 2010 Three-phase cross-cloud federation model: the cloud SSO authentication. In *2010 2nd Int. Conf. on Advances in Future Internet (AFIN)*, pp. 94–101. IEEE.
4. Zhang Z, Zhang X. 2009 Realization of open cloud computing federation based on mobile agent. In *IEEE Int. Conf. on Intelligent Computing and Intelligent Systems, ICIS 3*, vol. 3, pp. 642–646. IEEE.
5. Yara P, Ramachandran R, Balasubramanian G, Muthuswamy K, Chandrasekar D. 2009 Global software development with cloud platforms. In *Proc. 3rd Int. Conf. on Software Engineering Approaches for Offshore and Outsourced Development, Zurich, Switzerland, 2–3 July 2009*, pp. 81–95. Berlin, Germany: Springer.
6. Yara P, Ramachandran R, Balasubramanian G, Muthuswamy K, Chandrasekar D. 2009 Global software development with cloud platforms. In *Software engineering approaches for offshore and outsourced development*, vol. 35 (eds O Gotel, M Joseph, B Meyer, W Aalst, J Mylopoulos, M Rosemann, MJ Shaw, C Szyperski). Lecture Notes in Business Information Processing, pp. 81–95. Berlin, Germany: Springer.
7. Zeng W, Zhao Y, Ou K, Song W. 2009 Research on cloud storage architecture and key technologies. In *Proc. 2nd Int. Conf. on Interaction Sciences: Information Technology, Culture and Human*, pp. 1044–1048.
8. Harmer T, Wright P, Cunningham C, Hawkins J, Perrott R. 2010 An application-centric model for cloud management. In *2010 6th World Congress on Services*, pp. 439–446.
9. Geddes, N. 2006 The National Grid Service of the UK. In *2nd IEEE Int. Conf. on e-Science and Grid Computing*. IEEE.
10. Krieger O, McGachey P, Kanevsky A. 2010 Enabling a marketplace of clouds: VMware's vCloud director. *ACM SIGOPS Operating Syst. Rev.* **44**, 103–114. (doi:10.1145/1899928.1899942)
11. Fontán J, Vázquez T, Gonzalez L, Montero RS, Llorente IM. 2008 OpenNEbula: the open source virtual machine manager for cluster computing. In *Open source grid and cluster software conference: book of abstracts*.

12. Keahey K. 2009 Nimbus: open source infrastructure-as-a-service cloud computing software. In *Workshop on Adapting Applications AND Computing Services to Multi-core and Virtualization.* CERN.
13. OpenStack. See http://www.openstack.org/.
14. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. 2009 The Eucalyptus open-source cloud-computing system. In *Proc. 2009 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, pp. 124–131. IEEE.
15. Amazon Elastic Compute Cloud (Amazon EC2). See http://aws.amazon.com/ec2.
16. Wardley S, Goyer E, Barcet N. 2009 Ubuntu enterprise cloud architecture. Technical White Paper. Canonical.
17. Eucalyptus documentation. See http://open.eucalyptus.com/wiki/EucalyptusInstallation_v2.0.
18. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. 2007 KVM: the Linux virtual machine monitor. In *Proc. Linux Symp.*, pp. 225–230.
19. Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. 2003 Xen and the art of virtualization. In *Proc. 19th ACM Symp. Operating Systems Principles*, pp. 164–177.
20. UEC Cloud installation requirements. See https://help.ubuntu.com/community/UEC/CDInstall.
21. Intel White paper on UEC installation. See http://www.intel.com/Assets/PDF/general/icb_ra_cloud_computing_canonical_ubuntu.pdf.
22. Thain D, Tannenbaum T, Livny M, Berman F, Fox G, Hey T. 2002 Condor and the grid. In *Grid computing: making the global infrastructure a reality*, pp. 299–335. London, UK: Wiley.
23. Pronschinske M. 2009 Hybridfox: Elasticfox for Eucalyptus. Cloud Zone. See http://architects. dzone.com/news/hybridfox-elasticfox.
24. Massie M, Chun B, Culler D. 2004 The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* **30**, 817–840. (doi:10.1016/j.parco.2004.04.001)
25. Imamagic E, Dobrenic D. 2007 Grid infrastructure monitoring system based on Nagios. In *Proc. 2007 Workshop on Grid Monitoring*, June 2007, pp. 23–28. ACM.
26. OGF Usage Record Standard. See http://www.ogf.org/documents/GFD.98.pdf.
27. Weeks K. 2007 The National Grid Service user accounting system. In *Proc. UK e-Science All Hands Meeting, Nottingham, UK, 10–13 September 2007.*
28. Gluster File System. See http://www.gluster.org.
29. Square kilometre array. See http://www.skatelescope.org.
30. Buildbot. See http://trac.buildbot.net/.
31. VMWare ESXi. See http://www.vmware.com/products/vsphere-hypervisor/overview.html.