Monitoring Services in a Federated Cloud - The RESERVOIR Experience

Stuart Clayman, Giovanni Toffetti, Alex Galis, Clovis Chapman†
Dept of Electronic Engineering, Dept of Computer Science†, University College London
sclayman@ee.ucl.ac.uk, g.toffetti@ee.ucl.ac.uk, a.galis@ee.ucl.ac.uk, c.chapman@cs.ucl.ac.uk

ABSTRACT

This chapter presents the need, the requirements, and a design for a monitoring system that is suitable for supporting the operations and management of a Federated Cloud environment. We discuss these issues within the context of the RESERVOIR Service Cloud computing project. We first present the RESERVOIR architecture itself, then we introduce the issues of service monitoring in a federated environment, together with the specific solutions that have been devised for RESERVOIR. We end with a review of our experience in this area by showing a use-case application executing on RESERVOIR, which is responsible for the computational prediction of organic crystal structures.

INTRODUCTION

The emerging Cloud computing paradigm (Carr, 2008)(Wallis, 2008)(M. Armbrust et al., 2009) for hosting Internet-based services in virtualized environments, as exemplified by the Amazon Elastic Compute Cloud (EC2) or Google's AppEngine, aims to facilitate the creation of innovative Internet scale services without worrying about the computational infrastructure needed to support them. At present, no single hosting company can create a seemingly infinite infrastructure capable of serving the increasing number of on-line services, each having massive amounts of users and access at all times, from all locations. To cater to the needs of service creators, it is inevitable that the Service Cloud is going to be composed of a federation of sites from various infrastructure providers. Only by partnering and federating with each other, can infrastructure providers take advantage of the diversity factor and achieve the economies of scale needed to provide a seemingly infinite compute utility.

Service Clouds are just the latest incarnation of a concept that has been around since the 1960's, namely the manifestation of a general-purpose public computing utility. Throughout the history of computing we have seen such utilities appear in one form or another. Even though some success stories exist, such as in in the area of high performance scientific computing, where Grid computing made significant progress over the past decade, none of these attempts materialized into a true general purpose compute utility that is accessible by anyone, at any time, from anywhere. Now however, the advent of new approaches utilizing an always-on Internet and virtualization, has brought about system designs which will enable the desired progress. An example of such a system design is the RESERVOIR *Service Cloud*, which is described in the next section.

RESERVOIR

The RESERVOIR FP7 project (Rochwerger et al, 2009)(Rochwerger et al, 2009b)(Rochwerger et al, 2011) aims to support the emergence of Service-Oriented Computing as a new computing paradigm and to investigate the fundamental aspects of Service Clouds as a fundamental element of the Future Internet .

RESERVOIR is a Service Cloud which has a new and unique approach to Service-Oriented Cloud computing. In the RESERVOIR model there is a clear separation between *service providers* and *infrastructure providers*. Service providers are the entities that understand the needs of particular business and create and offer service applications to address those needs. Service providers do not need to own the

computational resources needed by these service applications, instead, they lease resources from an infrastructure provider.

The infrastructure provider owns and leases out sections of a computing cluster, which supplies the service provider with a finite pool of computational resources. The cluster is presented as a Service Cloud site which is capable of allocating resources to many service providers at the same time. Through federation agreements, multiple infrastructure providers can factor together all of their compute resources thus offering a seemingly infinite resource pool for their customers - the service providers.

The high-level objective of RESERVOIR is to significantly increase the effectiveness of the compute and service utility model thus enabling the deployment of complex services on a Service Cloud that spans infrastructure providers and even geographies, while ensuring QoS and security guarantees. In doing so, RESERVOIR provides a foundation where resources and services are transparently and flexibly provisioned and managed like utilities.

RESERVOIR ARCHITECTURE

The essence of the RESERVOIR Service Cloud is to effectively manage a service specified as a collection of virtual execution environments (VEEs). A VEE is an abstraction representing both virtual machines running on a generic hypervisor infrastructure, as well as any application component that can be run (and/or migrated) on a leased infrastructure (e.g., Web applications on Google's App Engine, a Java based OSGi bundle). A Service Cloud, such as RESERVOIR, operates by acting as a platform for running virtualized applications in VEEs, which have been deployed on behalf of a service provider.

The service provider defines the details and requirements of the application in a *Service Definition Manifest*. This is done by specifying which virtual machine images are required to run, as well as specifications for (i) Elasticity Rules or performance objectives, which determine how the application will scale across a Cloud, and (ii) Service Level Agreement (SLA) Rules, which determine how and if the Cloud site is providing the right level of service to the application. Within each Service Cloud site there is a Service Manager (SM) and a VEE Manager (VEEM) which together provide all the necessary management functionality for both the services and the infrastructure. These management components of a Cloud system are shown in Figure 1 and are presented in more detail.

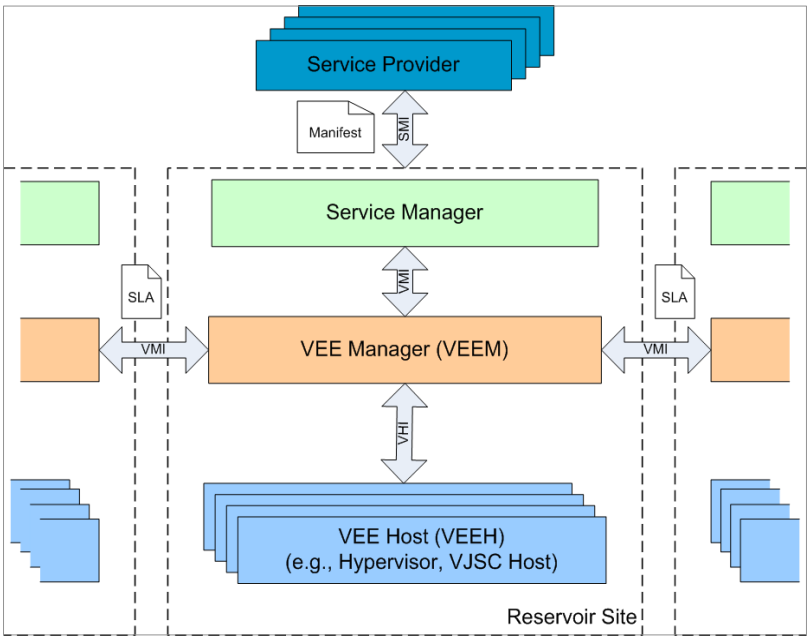

Figure 1: RESERVOIR Service Cloud Architecture

The *Service Manager* (SM) is the component responsible for accepting the *Service Definition Manifest* and the raw *VEE images* from the service provider. It is then responsible for the instantiation of the service

application by requesting the creation and configuration of executable VEEs for each service component in the manifest. In addition, it is the Service Manager that is responsible for (i) evaluating and executing the elasticity rules and (ii) ensuring SLA compliance, by monitoring the execution of the service applications in real-time. Elasticity of a service is done by adjusting the application capacity, either by adding or removing service components and/or changing the resource requirements of a particular component according to the load and measurable application behaviour.

The *Virtual Execution Environment Manager* (VEEM) is the component responsible for the placement of VEEs into VEE hosts (VEEHs). The VEEM receives requests from the Service Manager to create VEEs, to adjust resources allocated to VEEs, and to also finds the best placement for these VEEs in order to satisfy a given set of constraints. The role of the VEEM is to optimize a site and its main task is to place and move the VEEs anywhere, even on remote sites, as long as the placement is done within the constraints set in the Manifest, including specifications of VEE affinity, VEE anti-affinity, security, and cost. In addition to serving local requests, the VEEM is the component in the system that is responsible for the migration of VEEs to and from remote sites. This is achieved by interacting with the VEEMs that manage other Clouds.

The *Virtual Execution Environment Host* (VEEH) is a resource that can host a certain type of VEEs. For example one type of a VEEH can be a physical machine with the Xen hypervisor (Barham et al., 2003) controlling it, whereas another type can be a machine with the KVM hypervisor (Kivity, 2007). In a Service Cloud hosting site there is likely to be a considerable number of VEEHs organised as a cluster.

These three main components of the Service Cloud architecture interact with each other using specific interfaces, namely SMI (service management interface), VMI (VEE management interface), and VHI (VEE host interface), within a site and also use the VMI interface for site-to-site federation via the VEEM. In Figure 1 the relationship between these components and interfaces is shown.

In the RESERVOIR platform, as can be seen in Figure 1, a Service Provider specifies the details and requirements of his application in the *Service Definition Manifest*. The Manifest also has the specifications of Elasticity Rules, which determine how the application will scale across the Cloud, and Service Level Agreement (SLA) objectives for the application as well as the infrastructure. The former specify which performance objectives should be attained by the service, the latter are used to determine if the platform is providing the right level of service to the application.

Federation and Networking

Apart from virtual execution environment requirements, the *Service Definition Manifest* also specifies the networking requirements of a service. In particular, the manifest can specify one or more private virtual networks called *Virtual Area Networks* (VANs), implemented as virtual Ethernet overlay services. Furthermore, the manifest specifies the public access points for the deployed service. Virtual execution environment interfaces can then be mapped to public IP addresses.

Public interfaces for a service are mapped to the available public IP addresses available at each compute Cloud, which allows users of the service to access it via the public IP address. However, virtual area networks (VANs) are implemented at the infrastructure level ensuring the VAN has separation, isolation, elasticity, and federation. The advantages of these attributes of the VAN are explained further:

- *separation* — the service network and the infrastructure network are kept separate. RESERVOIR seeks to reduce mutual dependency between the infrastructure and the services. A VAN of a service, offered as part of RESERVOIR, needs to be separated from the infrastructure used by an infrastructure provider, similarly to the manner in which a virtual execution environment is separated from the physical host.
- *isolation* — a VAN of one service is isolated from all VANs of other services. RESERVOIR seeks to isolate services such that possibly competing applications may securely share the infrastructure provider resources whilst being unaware of the other services. Isolated VAN services need to be offered side by side while sharing network resources of the infrastructure provider.
- *elasticity* — a VAN can grow or shrink as necessary. RESERVOIR seeks to offer an elastic and extendable environment so that application providers will be able to adjust the size of their application on demand. A VAN service needs to enable application elasticity.

- *federation* — a VAN can span over more than one Cloud provider. RESERVOIR seeks to form a federation of possibly competing infrastructure providers so that each provider offers an interchangeable pool of resources allowing service and resource migration without barriers. An interchangeable VAN service needs to be offered across administrative domains such that service providers would not be concerned by the identity of the infrastructure provider, the physical network used, or its configuration.

The VAN implementation of RESERVOIR provides the required virtual Ethernet overlay for each service. As a consequence of service elasticity and site management policies, some virtual machines belonging to a same service might be placed across different sites.

Figure 2 illustrates a simple federated scenario across two Service Clouds (Cloud A on the left and Cloud B on the right). Each site leases out compute resources to different service providers, in our example Cloud A is serving services1 and 2 while Cloud B is managing service 3.

When virtual execution environments are placed on different sites, (as for Service 2 in Figure 2), the VEEMs at each participating site keep the VEEs connected by spawning the appropriate VAN proxies so that VANs stay connected and VEEs remain unaware of their actual placement. There are some network issues such latency, jitter, and round trip time considerations that arise, but the important aspect is that connectivity is maintained across the federated domains. In a more complex scenario (such as, when more services and Service Clouds are participating in the federation) a single service can be scattered across several sites requiring multiple VAN proxies.

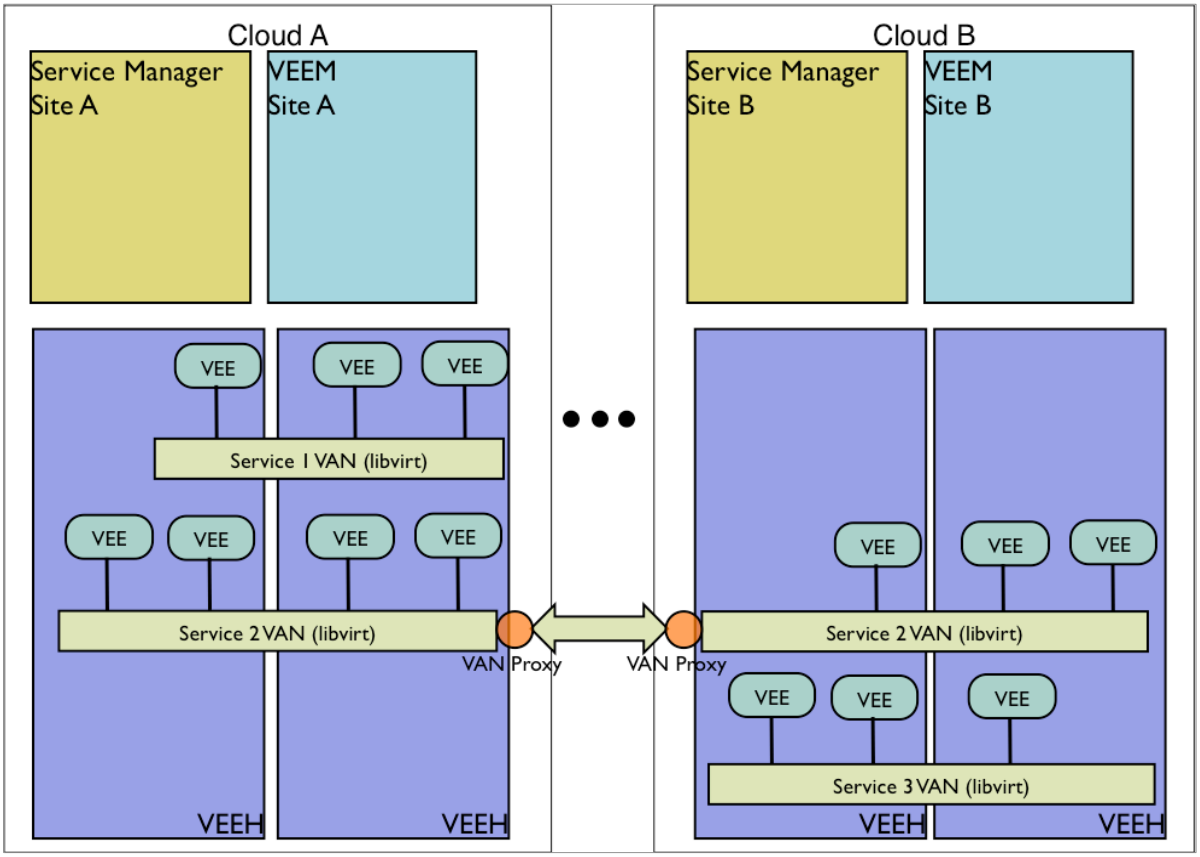

Figure 2: VANs and services running on two RESERVOIR sites

Once the VEEs are in place and the service is running, the Service Manager (SM) is responsible for managing all of the services on the Cloud. To undertake such management, a Cloud needs monitoring facilities.

MONITORING IN RESERVOIR

Monitoring is a fundamental aspect of a Service Cloud such as RESERVOIR because it is used both by the infrastructure itself and for service management. The monitoring system needs to be pervasive as:

- it is required by several components of the Service Cloud;
- it cuts across the layers of the Cloud system creating vertical paths; and
- it spans out across all the Service Clouds in a federation in order to link all the elements of a service.

For full operation of a RESERVOIR Service Cloud we observe that monitoring is a vital part of the full control loop that goes from the service management, through a control path, to the Probes which collect and send data, back to the service management which makes various decisions based on the data. The monitoring is a fundamental part of RESERVOIR as it allows the integration of components in all of the architectural layers.

The RESERVOIR monitoring system has a model of information consumers and information producers, connected by a monitoring data plane. The producers send measurements across the data plane, and these are read and processed by the consumers. In Figure 3 we show some of these producers and consumers, but this is not an exhaustive list.

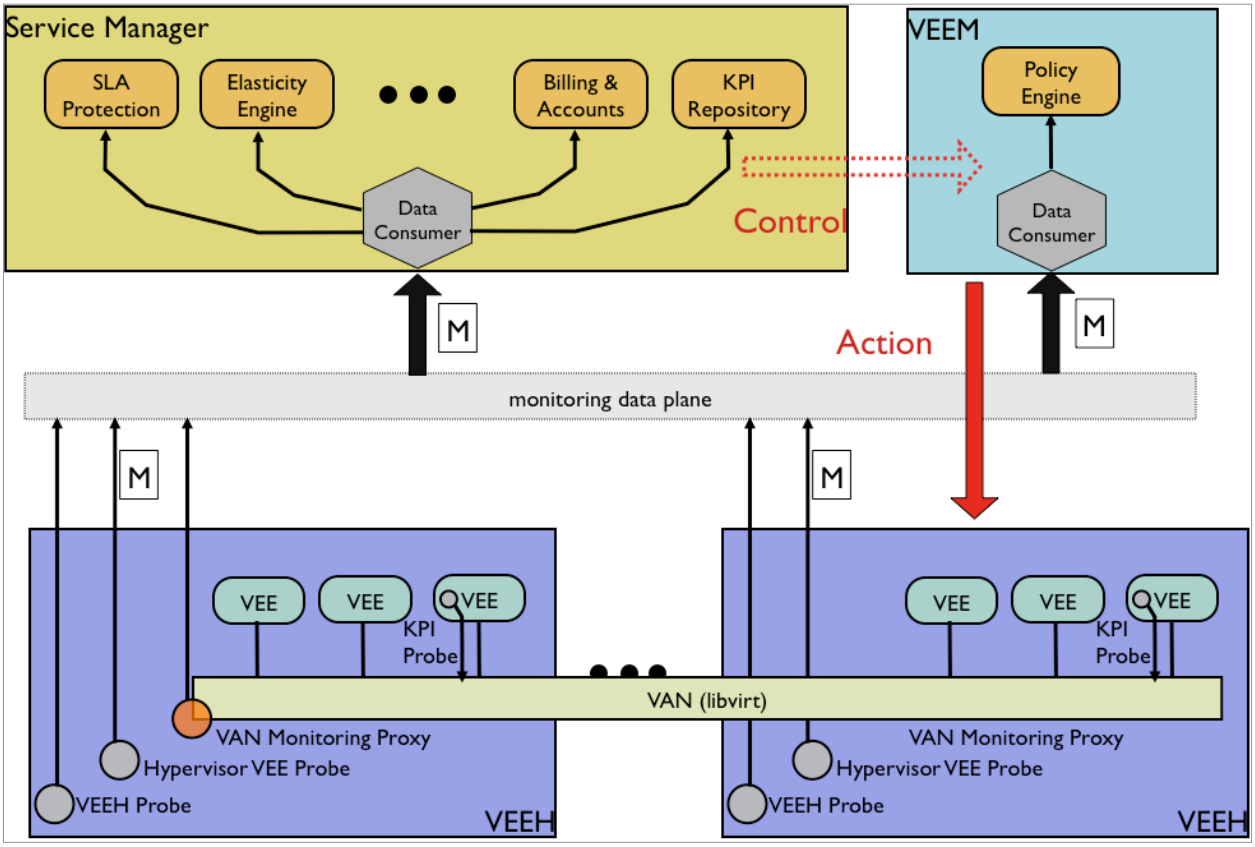

Figure 3: Information producers and consumers in a single RESERVOIR Cloud

Information Consumers

Several RESERVOIR management functionalities have monitoring requirements. Some of these are presented here:

- *The VEE placement* — within each RESERVOIR site, infrastructure resource utilization is monitored and the placement of VEEs is constantly updated to achieve optimal resource utilization according to policies set by the infrastructure provider. This is managed by Policy Engine of the VEEM;

- *Service billing* — each infrastructure provider needs to collect resource (e.g., CPU, disk, memory, network) utilization over time, for each virtual execution environment of each service, in order to support its billing policy. This is managed by the Billing and Accounts component of the Service Manager;
- *Service elasticity* — the execution of the service applications is monitored and their capacity is constantly adjusted to meet the requirements specified in the service definition manifest. These on-going optimizations are done without human intervention by the management components in the Service Manager. There are several components, including the Elasticity Engine and the SLA Protection Engine, which dynamically evaluate elasticity and SLA rules and implement their actions in order to maintain the application running effectively. For these rules to be evaluated, a Service Cloud needs a monitoring sub-system which sends measurement data, collected from the whole architecture, to be fed into the rules.
- *Infrastructural SLA compliance* — service providers require reports on resource provisioning to assess compliance to SLAs by infrastructural providers;
- *Access control* — infrastructure providers might use over-booking mechanisms to assign infrastructural resources to services based on a probabilistic model of resource usage derived from historical service traces. Hence, data about required VEEs and elasticity behaviour over time need to be stored and analysed.
- *Inter-site billing* — when virtual execution environments are migrated or spawned on a site belonging to a different service provider, billing across federated sites has to be supported according to federation agreements (e.g., billing on resource usage, or fixed amount)

For illustrative purposes, we gave just a high level overview of some of them.

Information Producers

Measurement data in RESERVOIR can come from the following sources (also depicted in Figure 3):

- raw data gathered from probes in the underlying infrastructure (VEEH). For example physical resource usage for each VEEH (VEEH Probe in Figure 3);
- raw data gathered from probes attached to the virtual machines (VEE). Used to monitor the amount of physical resources assigned to each VEE over time, mainly for billing (Hypervisor VEE Probe);
- data gathered from probes embedded in the application (application-specific probes), or data which is the combination of the raw data into composed Key Performance Indicators (KPI Probe through a VAN Monitoring Proxy);
- data derived from the analysis of historical raw data and KPI data held in a data store (from the KPI Repository of the Service Manager in Figure 3)

RESERVOIR enforces a clear distinction between monitoring data coming from the infrastructure (VEEH Probes, Hypervisor Probes) and monitoring data coming from services (KPI probes). Apart from being information of different nature, the former being Cloud-specific and the latter being service/application-specific, another important reason for this choice is that monitoring data coming from inside the application should be as small in size and as refined as possible.

The rationale behind this choice is that each application will have a very specific behaviour that is in principle known only to the service provider. Therefore, instead on burdening the infrastructure provider with the responsibility of collecting and manipulating a vast amount of low-informative raw monitoring data, the design choice is to leave the service providers with complete freedom in the ways of collecting, aggregating, cleaning, and manipulating all the raw monitoring data they will see most fit for their application. To get the service specific monitoring data out of the service and into the environment of the Cloud management, there is monitoring infrastructure “on the outside” of a service, called the *VAN Monitoring Proxy*, which expects the internal KPI Probes of a service to publish highly aggregated, highly informative performance indicators to be used for triggering automated service elasticity decisions.

RESERVOIR provides a reference framework for service providers to implement their own application-specific probes. Once a probe is instantiated according to the framework, the data it produces is published on the VAN and sent to a specific multicast address. A monitoring proxy intercepts the published information in the VAN and forwards it to the monitoring data plane for the local data consumers.

In a large distributed system, such as a RESERVOIR Service Cloud, there may be hundreds or thousands of measurement probes that can generate data. It would not be effective to have all of these probes sending data all of the time, so a mechanism is needed that controls and manages the relevant probes. Therefore, a Service Cloud requires a monitoring system that has a minimal runtime footprint and is not intrusive, so as not to adversely affect the performance of the Cloud itself or the running service applications. As a consequence, we need to ensure that components such as the VEEM, and Service Manager elements such as the Elasticity Rules Engine and the SLA Rules Engine only receive data that is of relevance and needed at a time.

Federated Monitoring

When Service Clouds are federated to accept each other's workload there needs to be a consideration of how monitoring will behave in the presence of the federated VEEs. Monitoring needs to continue to work correctly and reliably when a service executes across federated Clouds. When some VEEs are migrated to another Cloud, the monitoring data distribution mechanism will need to cross Cloud boundaries. It is essential that the interfaces and formats between Clouds be standardised in order that federation monitoring to work in heterogeneous environments. This will ensure that that the monitoring data for all the VEEs of a service will be connected, whether locally or remotely.

Monitoring in a Service Cloud presents us with some interesting issues. Although the Service Cloud infrastructure is a distributed system, it is structured in a very particular way, with one large Grid of machines acting as one Cloud. Most of the monitoring data stays within the site, as all of the consumers are within the site. The exception is for federated VEEs. With many monitoring systems the sources and consumers are often distributed arbitrarily across a network, and so the paths of data flow and the patterns of interaction are different. Within a Service Cloud the pattern is more predictable, and so we need to design and build for this. In Figure 3 we can see how the monitoring for the VEEs in one service are connected by the data distribution mechanism within a single Cloud, and how these are connected together to form a single service.

When a VEE is migrated from one site to a remote federated site, the monitoring data from that VEE still needs to be collected by the Service Manager at the originating Cloud. However, by migrating to a remote site the originating site loses direct control of the VEE. Therefore, in order to ensure a continuous monitoring capability, some fundamental issues need to considered. They are:

- how does monitoring behave when a VEE is sent to a remote Cloud
- what operations need to be done in each of the Clouds to activate this federation
- what components and objects need to be created and/or managed when federation occurs.

As we have seen in the 'Federation and Networking' section, once VEEs have crossed another Cloud's boundaries, the Ethernet overlay of the RESERVOIR VAN has the responsibility of instantiating the appropriate proxies so that VEEs belonging to the same service still have the perception of a single VAN spanning different Clouds. This connection between VAN parts on each Cloud is shown as the *VAN Segment* in Figure 4. If the service spans over 3 Clouds then there will be 3 Service VAN parts and 2 VAN Segments which comprise the whole service VAN.

Since there is a connection in place for each service, monitoring measurements from application-specific probes (KPI probes), on any VEE on any Cloud, will still reach the VAN Monitoring Proxy at the originating site. These measurements will be delivered to the originating site monitoring consumers. There is no need to provide any other mechanism to route service/application monitoring information for application-specific probes. Furthermore, VAN isolation provides guarantees that service/application-specific monitoring information will only be available at the originating site Service Manager.

A different solution has to be taken for infrastructural monitoring information. One group of measurements (i.e., messages from VEEH Probes) are not intended to leave the Cloud managing each VEEH. In fact, monitoring information on the physical machines of a specific Cloud is needed for the Cloud management (e.g., placement, anomaly detection) and generally kept hidden from the outside. Data from Hypervisor VEE Probes, on the other hand, needs to be received both by the local Cloud Service Manager (e.g., for inter-Cloud billing) as well as by the Service Manager at VEE originating site (e.g., for Service Provider billing). As a consequence, the monitoring data plane has a per-service segment between federated sites, in order to transmit this monitoring data. This connection between monitoring data plane parts on each Cloud is shown as the *Per-Service Segment* in Figure 4.

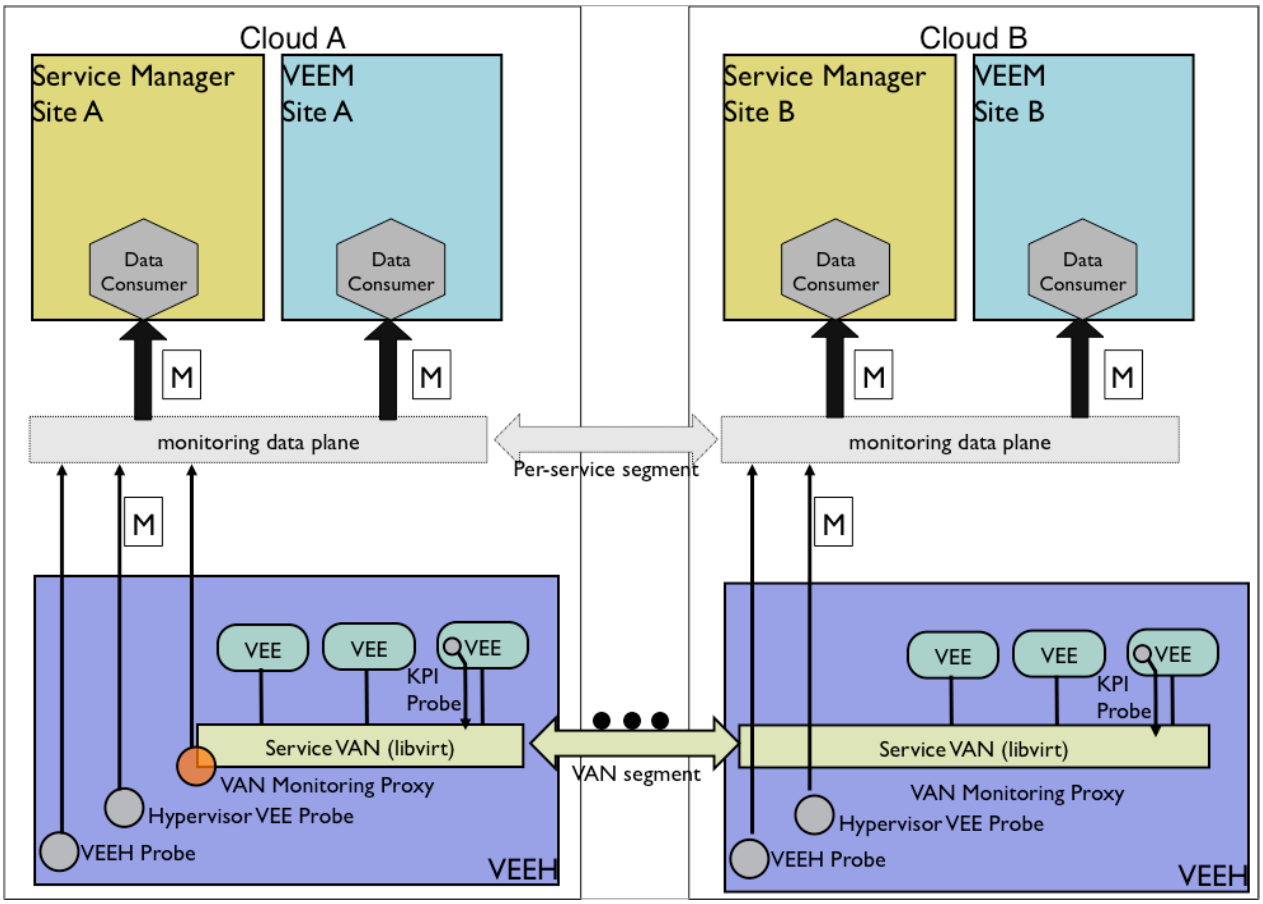

Figure 4: Monitoring across federated RESERVOIR Clouds

These Per-Service Segments of the monitoring data plane are setup and shutdown, on-demand, as VEEs migrate from one Cloud to another. The setup is done when the first VEE of a service is migrated to a federated Cloud, and the shutdown is done when the last VEE of a service is migrated off a federated Cloud. This allows the routing of Hypervisor Probe measurements to be delivered to the appropriate consumers.

When the first VEE of a service is about to be migrated across sites, a Federation Manager instantiates gateways for each service. A service gateway connects to the internal monitoring data plane of a RESERVOIR Cloud and forwards the relevant information to a matching gateway on the destination Cloud. In Figure 5 we see 3 such service gateways, for service A, service B, and service C. Only monitoring data from service A VEEs crosses the service segment through the service A gateways. The same process applies for service B and service C.

The service gateways are under the control of the Federation Manager, which itself is controlled and managed by the VEEM. As service gateways are under administrative control of the Cloud that instantiates them, this enables:

- complete control over the information that is chosen to be forwarded over the gateway, as the VEEM and Federation Manager can adjust the settings of the gateway;

- gateways to be adaptable and implemented according to each Cloud monitoring data plane technology. This allows for a Cloud to use one technology internally for the monitoring data plane, and use a different technology for the Cloud-to-Cloud transmission (e.g., multicast inside the Cloud and JMS over the gateway). Furthermore, two Clouds may have entirely different technologies for their monitoring data plane, but the gateways will provide the connection between them.

When VEEs are undeployed or migrated back to their originating site, and there are no more VEEs for a service still running on a Cloud, the federation managers on each Cloud have the responsibility of tearing down the appropriate service gateways.

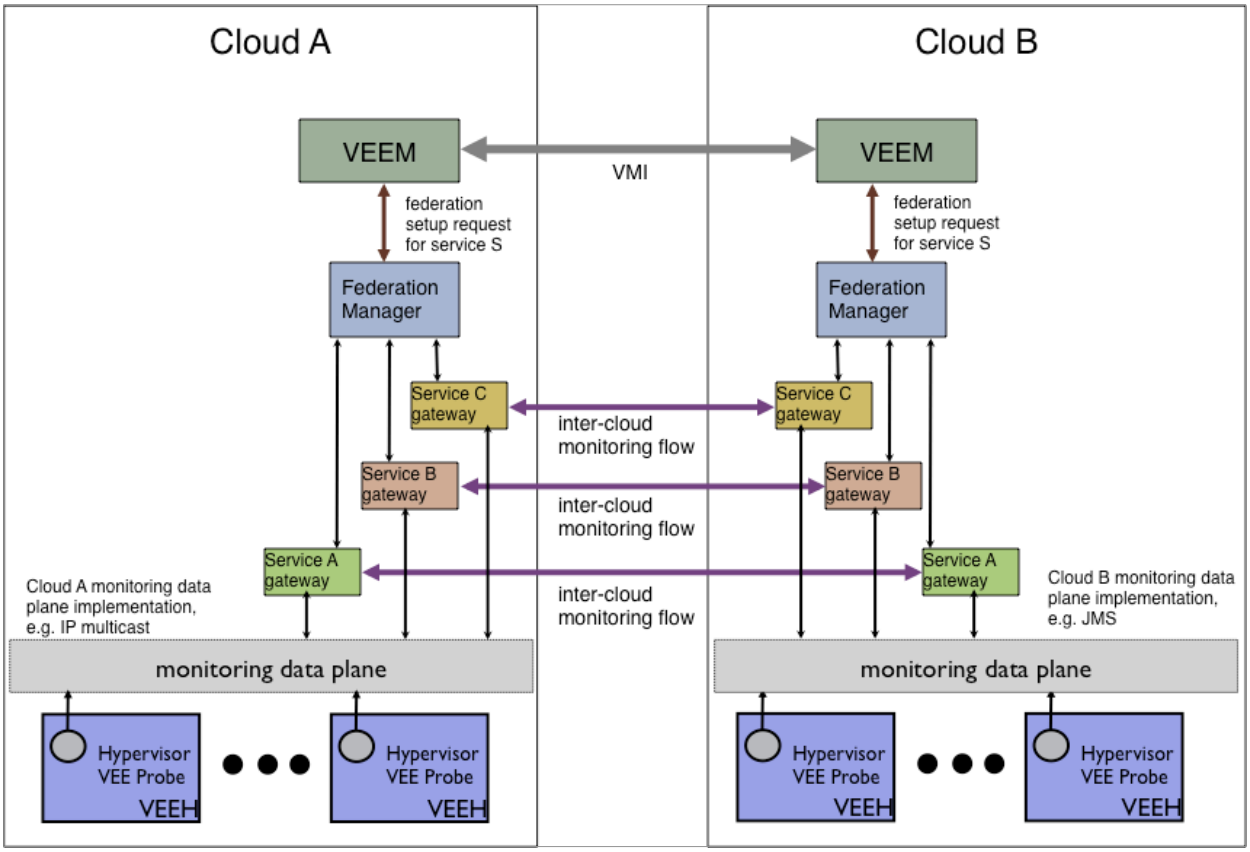

Figure 5: Federated monitoring management

The main requirements and issues regarding monitoring in a federated Cloud environment have been presented. In the next section we describe the design and implementation of a monitoring system that meets these requirements.

DESIGN OF A MONITORING SYSTEM

Existing monitoring systems such as Ganglia (Massie, Chun, & Culler, 2003), Nagios (Nagios), MonaLisa (Newman, Legrand, Galvez, Voicu, & Cirstoiu, 2003), and GridICE (Andreozzi et al., 2005) have addressed monitoring of large distributed systems, but they have not addressed the rapidly changing and dynamic infrastructure seen in Service Clouds.

There is a cloud monitoring system called Amazon CloudWatch (CloudWatch), that is available only through the Amazon AWS Cloud system. Amazon CloudWatch monitors AWS resources such as Amazon EC2 and Amazon RDS DB instances, and it can also monitor custom metrics generated by a customer's applications and services. It can be seen that its main goal is to provide data to the customer of the cloud about a running service, rather than providing data for infrastructure management.

Furthermore, there are two Cloud related projects that have also devised monitoring systems suitable for their needs. One is from the IRMOS project and the other is from the VISION project. The goal of the

IRMOS project (Irmos) is to design, develop, and validate Cloud solutions which will allow the adoption of interactive real-time applications, and especially multimedia applications, enabling a rich set of attributes with efficient integration into cloud infrastructures. The IRMOS monitoring system uses a two-layer approach which consists of six components. The goal of the VISION project (Vision) is the setup and deployment of virtual data and storage services on demand, across disparate administrative domains, providing QoS and security guarantees. Its monitoring system design is very extensive and adaptable. At this time, both projects are still on-going and it is too early to determine any conclusions on their monitoring systems.

A monitoring system for Clouds needs to be designed and built to be fit for the purpose of both infrastructure and service monitoring. It needs to be for the whole of infrastructure and service management, and so it should cover SLA compliance, elasticity, QoS, etc. It is important to recognise that it is the monitoring mechanism that closes the loop from the initial deployment, through execution, and back to the Service Manager (as shown in Figure 3). The monitoring system is there to gather data from all the components within a Cloud architecture, and so monitoring is a fundamental aspect of a Service Cloud that is used by the infrastructure and for service management.

The monitoring system for a Service Cloud needs to feed data into the Service Manager so that it can manage the services deployed on the Cloud. Over time, it is expected that the management capabilities of the Service Manager will expand to include new functions. As a consequence we need the monitoring system to be adaptable, flexible, and extensible in order to support the expanding functionality.

To address all of the requirements and functionality of the Service Cloud environment, we have determined that the main features for monitoring that need to be considered are:

- *scalability* — to ensure that the monitoring can cope with a large number of probes
- *elasticity* — so that virtual resources created and destroyed by expanding and contracting networks are monitored correctly
- *migration* — so that any virtual resource which moves from one physical host to another is monitored correctly
- *adaptability* — so that the monitoring framework can adapt to varying computational and network loads in order not to be invasive
- *autonomic* — so that the monitoring framework can keep running without intervention and reconfiguration
- *federation* — so that any virtual resource which resides on another domain is monitored correctly
- *isolation* — so that monitoring of VEEs from different services are not mixed and are not visible to other services

To establish such features in a monitoring framework requires careful architecture and design. The following section presents the Lattice Monitoring Framework which we have designed and built for the purpose of monitoring dynamic environments such as Service Clouds. Further details of the use of Lattice can be found in (Clayman et al., 2010) and (Clayman, Galis, & Mamatas, 2010).

In many systems, probes are used to collect data for system management (Cooke et al., 2003) (Massie et al., 2003). In this regard, Lattice also relies on probes. However, to increase the power and flexibility of the monitoring we introduce the concept of a data source. A data source represents an interaction and control point within the system that encapsulates one or more probes. A probe sends a well defined set of attributes and values to a data consumer at a predefined interval.

The goal for the monitoring system is to have fully dynamic data sources, in which each one can have multiple probes, with each probe returning its own data. The data sources will be able to turn on and turn off probes, or change their sending rate dynamically at run time. A further useful facility for a data source is the ability to add new probes to a data source at run-time. By using this approach we will be able to instrument components of the system without having to restart them in order to get new information.

It is also beneficial to interface with existing frameworks in order to collect data for Lattice. We would need these frameworks to fit in with the concept of data source and probe, and so they can be encapsulated with the relevant adapter in the implementation.

Such an approach for data sources and probes is important because in many systems that need monitoring , the Service Management needs to grow and adapt for new management requirements over time. If the monitoring system is a fixed point then Service Management will be limited.

To meet all the criteria outlined requires careful architecture and design. Many monitoring systems rely on simple data transmission. From a design point of view, this approach is successful, although, we have found it is better if the monitoring framework design encapsulates separate planes for data, for meta-data, and for control. This allows us to build a system that has the desired behaviour and meets the requirements.

In Lattice, the separate planes for connecting the monitoring framework are:

- the *data plane* — for distributing measurements from the Probes and Data Sources to the consumers. This is the same monitoring data plane shown in Figures 3, 4, and 5.
- the *control plane* — for distributing control messages to the Data Sources and the Probes.
- the *information plane* — which holds all the meta-data relating to measurements sent by Data Sources and Probes.

These three planes are shown in Figure 6, together with the Data Sources and Probes, Data Consumers, and a Regulator.

In a running system there will be multiple Data Sources, multiple Data Consumers, but only one Regulator. The regulator's job is to ensure that the monitoring system does not flood the network or overload any applications.

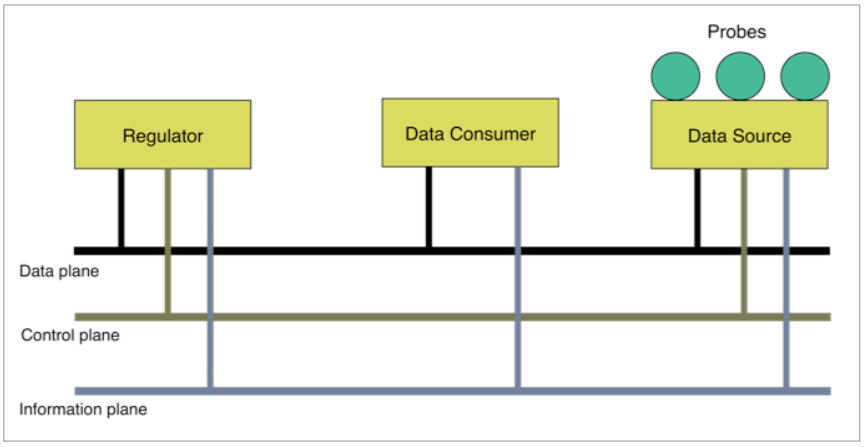

Figure 6: Multiple Planes

Data Source

A Data Source can manage several Probes, and has plugins for the monitoring data plane, the control plane, and the information plane, so that it is possible to change the implementation of each plane easily and independently of the other planes. This will allow users to choose the best solution for different setup scenarios. We have a control interface so that Data Sources can be controlled from a Service Manager, via the control plane, and Probes can also be controlled via the Data Source. We can then develop control strategies so that it is possible to manage the lifecycle of a Probe from a Service Manager rather than having everything programmed-in.

As there could be hundreds or thousands of probes in a Lattice system, it is important that each probe has a unique identity. Without an identity, it is not possible to identify individual probes. Using the identity it is possible for the Data Source to address the probe in order to turn it on, turn it off, change its rate of sending, or find its status. It is the probe's identity that also allows the combination of its data with other probe's data to create the complex information.

Probes

In many systems, the Probes collect data at a given data rate, and transmit measurements immediately, at exactly the same data rate. In Lattice we can decouple the collection rate and the transmission rate in order to implement strategies which aid in efficiency.

The collection of measurement data is a fundamental aspect of a Probe. The collection strategies will be:

- at data rate, which collects some measurement data at a regular data rate.
- on event, where a measurement is not collected at a specified rate, but is passed to the probe as an event from another entity.

The transmission strategies are:

- at data rate, this is the main strategy, which transmits measurements at a regular data rate.
- on change, where a measurement is only transmitted when the data that is read is different from the previous data snapshot. This can be elaborated so that only a specific set of attributes are included when determining if there has been a change.
- filtering, where a measurement is only transmitted if the filter passes the value. Examples of such filtering strategies are:
  - above threshold, where a measurement is only transmitted when an attribute of the data that is read is above a specified threshold. Otherwise, nothing is transmitted.
  - below threshold, where a measurement is only transmitted when an attribute of the data that is read is below a specified threshold. Otherwise, nothing is transmitted.
  - in band, where a measurement is only transmitted when an attribute of the data that is read is between an upper bound and a lower bound. Otherwise, nothing is transmitted.
  - out of band, where a measurement is only transmitted when an attribute of the data that is read is above an upper bound or below a lower bound. Otherwise, nothing is transmitted.

  although any kind of filter can be defined.

For a probe to be part of Lattice we can either write the probe from scratch or use existing sensors and instrumentation and adapt them for Lattice. The probes can be implemented in various ways to get the data they need to send. They can:

- read the data directly from the relevant place
- be an adaptor for an existing instrument by wrapping the existing one
- act as a bridge for an existing instrument by allowing the existing one to send it data.

in order to gather the required information.

Probe Data Dictionary

One of the important aspects of this monitoring design is the specification of a Data Dictionary for each probe. The Data Dictionary defines the attributes as the names, the types and the units of the measurements that the probe will be sending out. This is important because the consumers of the data can collect this information in order to determine what will be received. At present many monitoring systems have fixed data sets, with a the format of measurements being pre-defined. The advantage here is that as new probes are to be added to the system or embedded in an application, it will be possible to introspect what is being

measured. This is important in a Service Cloud system such as RESERVOIR, because many of the Probes will not be known in advance.

The measurements that are sent will have value fields that relate directly to the data dictionary. To determine which field is which, the consumer can lookup in the data dictionary to elaborate the full attribute value set.

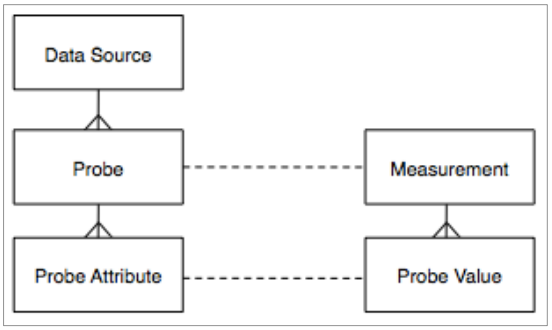

Figure 7: Relationship of Probe to Measurement

Measurements

The actual measurements that get sent from a probe will contain the *attribute-value* fields together with a type, a timestamp, plus some identification fields. The attribute-values contain the information the probe wants to send, the type indicates what kind of data it is, and the timestamp has the time at which the data was collected. The identification fields are used to determine for which component or which service, and from which probe this data has arrived from.

When using Lattice, if there are multiple components which need monitoring and there are multiple running services and any of these can have multiple probes, then the consumer of the data will be able to differentiate the arriving data into the relevant streams of measurements.

Data Plane

In order to distribute the measurements collected by the monitoring system, we use the Data Plane. For most Data Plane uses, we need a mechanism that allows for multiple submitters and multiple receivers of data without having vast numbers of network connections. For example, having many TCP connections from each producer to all of the consumers of the data for that producer would create a combinatorial explosion of connections. Solutions to this include IP multicast, Event Service Bus, or publish/subscribe mechanism. In each of these, a producer of data only needs to send one copy of a measurement (in a packet) onto the network, and each of the consumers will be able to collect the same packet of data concurrently from the network.

In order to avoid a reliance on any particular networking technology, there is a Data Plane plugin capability that allows the actual implementation for data distribution to be set at run time. Such an approach works well for Clouds and Grids because we have both multiple producers and multiple consumers.

One of the easiest solutions for the Data Plane is to use IP multicast because it has a very simple mechanism to distribution. However, IP multicast is a UDP based networking technology, and as such it has very similar attributes in terms of reliability and packet loss. It is possible to consider a more substantial framework which has higher reliability and more flexibility for the Data Plane as well.

Control Plane

In many monitoring systems, the probes are reporting their status at a fixed data rate and are always sending data onto the network. Such an approach is not scalable and does not fit with the design goals of Lattice. The

monitoring system for Lattice requires both scalability and flexibility in order to operate effectively. Consequently, we need to have a level of control over all of the Data Sources and all of the Probes.

We have defined operations that allow individual Probes to be turned on and turned off independently of any other Probe. Furthermore, any Probe can have its data rate changed on-the-fly at run-time. It is by using these capabilities that we can achieve the scalability and flexibility we need from the monitoring system.

We have defined the Control Plane to support all of the above functionality. To enact this we have devised the following artefacts:

- a control protocol which sends the control messages,
- a controller-manager which decides which Probes to interact with,
- a policy engine which takes a global view of all the monitoring information and interacts with the controller-manager

Once these are built we have dynamic and autonomic control over the monitoring.

Information Plane

The Information Plane allows interaction with the Information Model for the monitoring framework. It holds all of the data about Data Sources, Probes, and Probe Data Dictionaries present in a running system. As Measurements are sent with only the values for the current reading, the meta-data needs to be kept for lookup purposes. By having this Information Plane, it allows consumers of measurements to lookup the meaning of each of the fields.

Within Lattice, there are key lifecycle points where data is added to the information model. They are when a Data Source is activated, and when a Probe is activated. This data can be looked up as required.
For example, if a measurement has a probe ID of 4272, as seen earlier, then a Service Manager can interact with the Information Plane to extract the actual name of the probe.

In many older monitoring systems this information model is stored in a central repository, such as an LDAP server. Newer monitoring systems use a distributed approach to holding this data, with MonAlisa using JINI as its information model store. For Lattice, the Information Plane also relies on a plugin component. Currently, we have an implementation which uses a Distributed Hash Table.

A MONITORING USE CASE

We have seen the RESERVOIR architecture, the monitoring requirements for RESERVOIR, and the design for an implementation of a monitoring system, we now present a service that executes on a Cloud in order to highlight how the monitoring is essential and how services need to be adapted to work on the Cloud. In this section, we give an overview of a service that can take full advantage of a RESERVOIR Cloud.

The selected service is a Grid-based application responsible for the computational prediction of organic crystal structures from the chemical diagram (Emmerich, Butchart, Chen, Wassermann, & Price 2005). The application operates according to a pre-defined workflow involving multiple web-based services and Fortran programs. Up to 7200 executions of these programs may be required to run, as batch jobs, in both sequential and parallel form, to compute various subsets of the prediction. Web services are used to collect inputs from a user, coordinate the execution of the jobs, process and display results, and generally orchestrate the overall workflow. The actual execution of batch jobs is handled by Condor (Thain, Tannenbaum, & Livny, 2002), a job scheduling and resource management system, which maintains a queue of jobs and manages their parallel execution on multiple nodes of a cluster.

This case study provides many interesting challenges when deployed on a Cloud computing infrastructure such as RESERVOIR. Firstly, the application consists of a number of different components with very different resource requirements, which are to be managed jointly. Secondly, the resource requirements of the services will vary during the lifetime of the application. Indeed, as jobs are created, the number of cluster nodes required to execute them will vary. Our goal in relying upon a Cloud computing infrastructure will be to create a virtualised cluster, enabling the size of the cluster to dynamically grow and contract according to load.

The service setup is illustrated in Figure 8, where the three main types of service components can be distinguished:

- The *Orchestration Service* is a web based server responsible for managing the overall execution of the application. It presents an HTTP front end enabling users to trigger predictions from a web page, with various input parameters of their choice. The Business Process Execution Language (BPEL), is used to coordinate the overall execution of the polymorph search, relying on external services to generate batch jobs, submit the jobs for execution, process the results and trigger new computations if required.
- The *Grid Management Service* is responsible for coordinating the execution of batch jobs. It presents a web services interface for the submission of jobs. Requests are authenticated, processed and delegated to a Condor scheduler, which will maintain a queue of jobs and manage their execution on a collection of available remote execution nodes. It will match jobs to execution nodes according to the workload. Once a target node has been selected it will transfer input files over and remotely monitor the execution of the job.
- The *Condor Execution Service*, which runs the necessary daemons to act as a Condor execution node. These daemons will advertise the node as an available resource on which jobs can be run, receive job details from the scheduler and run the jobs as local processes. Each node runs only a single job at a time and upon completion of the job transfers the output back to the scheduler, and advertises itself as available.

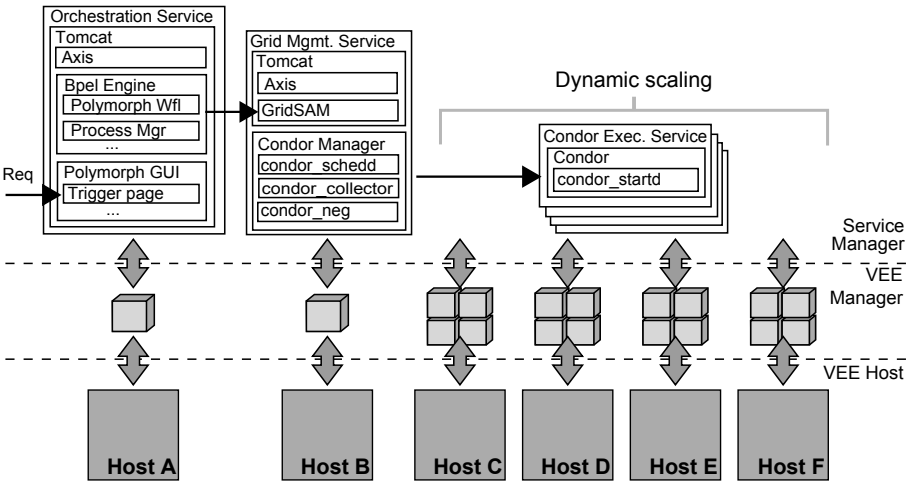

Figure 8: Service on RESERVOIR

Packaged as individual virtual execution environments, the three components are deployed on the RESERVOIR infrastructure. The associated manifest describes the capacity requirements of each component, including CPU and memory requirements, references to the image files, starting order (based on service components dependencies), elasticity rules and customisation parameters. For the run shown here, the Orchestration and Grid Management Services will be allocated a fixed set of resources, with only a single instance of each being required. The Condor execution service however will be replicated as necessary, in order to provide an appropriate cluster size for the parallel execution of multiple jobs.

Elasticity rules will tie the number of required Condor execution service instances to the number of jobs in queue as presented by the Condor scheduler. This enables RESERVOIR to dynamically deploy new execution service instances as the number of jobs awaiting execution increases. Similarly, as the number of jobs in the queue decreases it is no longer necessary to use the resources to maintain a large collection of execution nodes, and VEEs can be released accordingly.

The elasticity rules will refer to key performance indicators that are declared within the manifest. All components and KPIs are declared in the manifest. This enables the infrastructure to monitor KPI measurements being published by specific components and associate them to the declared rules. In this

particular instance, a monitoring Probe associated with the Grid Management Service will publish measurements under the `uk.ucl.condor.schedd.queuesize` qualified name every 30 seconds as integers.

When the Grid Management component is operational, the monitoring Probe, will begin the process of monitoring the queue length and transmit the number of jobs in the queue on a regular basis (every 30 seconds). These monitoring events will be recorded by the Service Manager to enforce elasticity rules. When conditions regarding the queue length are met, the Service Manager will request the deployment of an additional Condor Execution component instances. Similarly, when the number of jobs in queue falls below the selected threshold, it will request the deallocation of virtual instances.

An important metric to consider is that of resource usage. The goal of Cloud deployment is to reduce expenditure by allowing Service Providers to minimise over-provisioning. While the actual financial costs will be dependent on the business models employed by Cloud infrastructure providers, we can at the very least rely upon resource usage as an indicator of cost.

We compare resource usage obtained on RESERVOIR with that obtained in an environment with dedicated physical resources. The objective is to verify that there is a significant reduction in resource usage. The results illustrated in Figure 9 show the number of queued jobs plotted against the number of Condor execution instances deployed. Both charts show large increases in queued jobs as the first long running jobs complete and the larger sets are submitted. More importantly, the first chart represents the execution of the application in a dedicated environment and shows a set of 16 continuously allocated execution nodes. The second chart represents the execution of the application with RESERVOIR, shows the increase in the number of allocated nodes as jobs in queue increases, and a complete deallocation as these jobs complete.

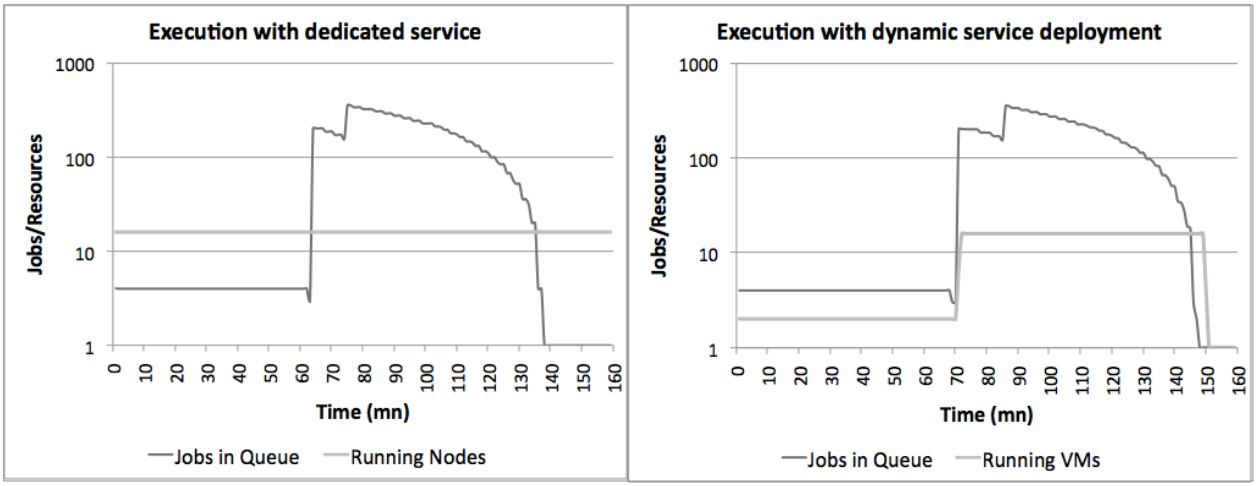

Figure 9: Service on RESERVOIR

There is little difference in execution times in the individual batches of jobs on either the dedicated or virtual RESERVOIR environment. A 10 minute increase of time, using RESERVOIR, can be constituted as reasonable considering the overall time frame of a search, which is well over 2 hours. This is particularly true as we consider the overall resource usage savings. Indeed as can be seen in table 1, with respect to execution nodes, the overall resource usage decreases by 34.46% by relying on service elasticity. This is because the totality of the execution nodes are not required for the initial bulk of the run, where only 2 jobs are to be run.

The overall resource usage obtained showing run times, nodes used, and the summary is shown in Table 1.

	Dedicated Environment	RESERVOIR Environment
Search run time (min)	143.41	153.66
Complete shutdown time (min)	No shutdown	159.11

Average execution nodes:		
- for run	16	10.49
- until shutdown	No shutdown	10.42
Percentage differences:		
- Resource usage saving		-34.46%
- Extra run time (jobs)		7.15%

Table 1: Service on RESERVOIR

The savings shown here are in the context of the run itself. If we consider the overall use of the application over the course of a randomly selected week on a fully dedicated environment where resources are continuously available, even more significant cost savings will exist. Based on cost savings obtained here, we have estimated that overall resource consumption would drop by 69.18%, due to the fact that searches are not run continuously; no searches were run on two days of the week, and searches, though of varying size, were run only over a portion of the day, leaving resources unused for considerable amounts of time. More details of this work can be found in (Chapman, Emmerich, Marquez, Clayman, & Galis, 2011).

FUTURE RESEARCH DIRECTIONS

There is much further work to do in the areas of service clouds and federated monitoring.

Within RESERVOIR, there is a full monitoring system lifecycle integration within the Service Manager, from the initial Service Definition Manifest presentation, through the first service deployment, and then the continual execution of the service. To support fully dynamic on-the-fly functionality these lifecycle and control functions will need to use a control plane of the monitoring framework. Using this control plane the Service Manager will be able to activate, deactivate, and set the data rate of probes, as necessary.

Furthermore, to ensure the continuous operation of the monitoring system, it needs to be self-aware of its operation and its effect on the whole RESERVOIR system. In order to make it this way, the monitoring system needs its own adaptable controller with policies which can control the whole of the monitoring system. All of these elements are considered as part of the further work. One main area of our future work will be to address the main features for monitoring in a virtualized network environment within the Lattice framework itself.

As we have shown, for full operation of a virtualized service cloud environment the monitoring is a vital part of the control loop. To ensure that this loop is effective we need to be confident that the measurements transmitted by probes are accurate. As such, we intend to investigate and evaluate the accuracy of the monitoring framework.

The mechanism of defining the measurements that probes will send using an information model is aligned with modelling approaches in many other areas. Consequently, we will examine how we can take information models for the service applications, and use this to create information models for the probes that will monitor these components. By doing this we could define and create the probes automatically from the service application’s model.

A further goal for the Lattice monitoring system is to have fully dynamic data sources, in which the data source implementation will have the capability for reprogramming the probes on-the-fly. In this way, it will be possible to make probes send new data if it is required. For example, they can send extra attributes as part of the measurement. Yet another useful facility for a data source will be the ability to add new probes to a data source at run-time. By using this approach we will be able to instrument components of the virtual network without having to restart them in order to get this new information.

CONCLUSIONS

Monitoring in Service Clouds, such as RESERVOIR, presents us with some interesting issues. Although the Service Cloud infrastructure is a distributed system, it is structured in a very particular way, with one large cluster of machines acting as one Cloud site. Most of the monitoring data stays within the site, as all of the consumers are within the site. The only exception to this is for federated VEEs. We see that the monitoring system is pervasive to a Service Cloud as:

- it is required by most of the components of the Service Cloud;

- it cuts across the layers of the Service Cloud creating vertical paths; and
- it spans out across all the Clouds in a federation in order to link all the VEEs of a service.

With many monitoring systems the sources and consumers are often distributed arbitrarily across a network, and so the paths of data flow and the patterns of interaction are different. Within a Service Cloud the pattern is more predictable, and so we have designed and built for this situation. In this chapter we have presented the issues that need to be addressed for a federated Cloud and we have shown how federated monitoring is designed and activated within RESERVOIR.

The use of the Lattice framework allowed us to build a monitoring infrastructure that collects, processes, and disseminates network and system information from/to the entities at real-time, acting as an enabler for service management functionality. We have seen that monitoring is a fundamental feature of any Service Cloud management system. We have also shown that Lattice can provide such monitoring for systems in which the virtualized resources are highly volatile as they can be started, stopped, or migrated at any time by the management in a federated environment.

We have shown that the implementation of these concepts is feasible and that a complete architecture can enable a Service Cloud infrastructure to realise significant savings in resource usage, with little impact on overall quality of service. Given that a Cloud can deliver a near similar quality of service as a dedicated computing resource, Clouds then have a substantial number of advantages. Firstly application providers do not need to embark on capital expenditure and instead can lease the infrastructure when they need it. Secondly, because the elasticity rules enable the application provider to flexibly expand and shrink their resource demands so they only pay the resources that they actually need. We have seen in the use-case example that the overall resource usage decreases by 34.46% by relying on service elasticity, as compared to a dedicated environment. Finally, the Cloud provider can plan their capacity more accurately because it knows the resource demands of the applications it provides.

Acknowledgment

This work is partially supported by the European Union through the RESERVOIR project of the 7th Framework Program. We would like to thank the members of the RESERVOIR consortium for their helpful comments. We also would like to thank Benny Rochwerger, Lars Larsson, Luis M. Vaquero, Luis Rodero Merino, Ruben S. Montero, Kenneth Nagin, Alessio Gambi, and Mario Bisignani for the technical discussions and support.

REFERENCES

Andreozzi, S., De Bortoli, N., Fantinel, S., Ghiselli, A., Rubini, G. L., Tortone, G., & Vistoli, M. C. (2005). GridICE: A monitoring service for Grid systems. *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 559–571.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2009). Above the Clouds: A Berkeley view of Cloud computing. *Technical Report UCB/EECS-2009-28*, EECS Department, University of California, Berkeley

Barham, P., Dragovic, B., Fraser, K., Hand, S., et al. (2003). Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177.

Carr, N. (2008). The Big Switch - Rewiring the World from Edison to Google. W. W. Norton.

Chapman, C., Emmerich, W., Marquez, F.G., Clayman, S., & Galis, A. (2011). Software Architecture Definition for On-demand Cloud Provisioning, *Special Issue of Cluster Computing*

Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero-Merino, L., Vaquero, L., Nagin, K., & Rochwerger, B. (2010). Monitoring Service Clouds in the Future Internet. In *Towards the Future Internet - Emerging Trends from European Research*, pages 115–126, Amsterdam, The Netherlands, The Netherlands, IOS Press.

Clayman, S., Galis, A., & Mamatas, L. (2010). Monitoring virtual networks with Lattice. In: *Management of Future Internet* - ManFI 2010, URL http://www.manfi.org/2010/

CloudWatch, Amazon CloudWatch, http://aws.amazon.com/cloudwatch/

Cooke, A., Gray, A. J. G., Ma, L., Nutt, W. et al., (2003). R-GMA: An information integration system for Grid monitoring. In *Proceedings of the 11th International Conference on Cooperative Information Systems*, pp. 462–481.

Emmerich, W., Butchart, B., Chen, L., Wassermann, B., & Price, S. L. (2005). Grid Service Orchestration using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3(3- 4):283–304.

Irmos, IRMOS project, http://www.irmosproject.eu/

Kivity, A. (2007). KVM: the linux virtual machine monitor," in *OLS '07*: The 2007 Ottawa Linux Symposium, pp. 225–230.

Massie, M. L., Chun, B. N., & Culler, D. E. (2003). The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, vol. 30.

Nagios, http://www.nagios.org/

Newman, H. Legrand, I., Galvez, P., Voicu, R., & Cirstoiu, C. (2003). MonALISA : A distributed monitoring service architecture. In *Proceedings of CHEP03*, La Jolla, California.

Rochwerger, B., Breitgand, D., Levy, E., Galis, A. et al., (2009). The RESERVOIR model and architecture for open federated Cloud computing. *IBM Journal of Research and Development*, vol. 53, no. 4.

Rochwerger, B., Galis, A., Levy, E., Caceres, J. A., Breitgand, D., Wolfsthal, Y., Llorente, I. M., Wusthoff, M., Montero, R. S., & Elmroth, E. (2009b). Management technologies and requirements for next generation service oriented infrastructures. In 11th *IFIP/IEEE International Symposium on Integrated Management*.

Rochwerger, B., Tordsson, J., Ragusa, C., Breitgand, D., Clayman, S., Epstein, A., Hadas, D., Levy, E., Loy, I., Maraschini, A., Massonet, P., Munoz, H., Nagin, K., Toffetti, G., & Villari, M. (2011). RESERVOIR - When one Cloud is not enough. In *IEEE Computing*.

Thain, D., Tannenbaum, T., & Livny, M., (2002). Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc.

Vision, VISION project, http://www.visioncloud.eu/

Wallis, P. (2008). Understanding Cloud computing, keystones and rivets., http://www.keystonesandrivets.com/kar/2008/02/Cloud-computing.html.

ADDITIONAL READING SECTION

4CaaSt, Building the PaaS Cloud of the Future, http://4caast.morfeo-project.org

4WARD, The 4WARD Project - http://www.4ward-project.eu/

AmazonWS, Amazon Web Services, http://aws.amazon.com/

Brewer, E.A. (2000) "Towards robust distributed systems". In Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, vol. 19, pp. 7-10,

Chapman, C., Emmerich, W., Marquez, F.G., Clayman, S., and Galis, A., (2010), *Elastic Service Management in Computational Clouds*, CloudMan 2010, IEEE/IFIP, Osaka, Japan, 19-23 April 2010.

CIMI, Cloud Infrastructure Management Interface, DMTF, http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.0b_0.pdf

Elmroth, E., Marquez, F.G. , Henriksson, D. , and Ferrera. D.P., (2009), *Accounting and billing for federated cloud infrastructures*, 8th Intl. Conf. Grid and Cooperative Computing, 268-275, 2009.

Jeery, K., and Burkhard, L., (2010), The Future of Cloud Computing: Opportunities For European Cloud Computing Beyond

Kang, C., and Wei-Min, Z.. (2009), *Cloud Computing: System Instances and Current Research*, Journal of Software, Vol.20, No.5, May 2009, pp.1337 1348

Legrand, I., Voicu, R. , Cirstoiu, C. , Grigoras, C. , Betev, L., Costan, A., *Monitoring and Control of Large Systems With MonALISA*, Communications of the ACM Vol. 52 No. 9, Pages 49-55.

McKinley, P.K., Samimi, F.A., Shapiro, J.K., Tang, C., (2005), *Service Clouds: A distributed infrastructure for composing autonomic communication services*. Technical Report, MSU-CSE-05-31, Department of Computer Science, Michigan State University, East Lansing, Michigan

Milojicic, D.S., Llorente, I.M., Montero. R.S., (2011), *OpenNebula: A Cloud Management Tool*. IEEE Internet Computing, 15(2):11-14, 2011.

Mejias, B., and Van Roy, P., (2010), *From Mini-clouds to Cloud Computing*. In Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW '10). IEEE Computer Society, Washington, DC, USA, 234-238.

OCCI, Open Cloud Computing Interface, OGF

OpenNebula, The Open Source Toolkit for Data Center Virtualization (OpenNebula), http://opennebula.org/

OVF, Open Virtualization Format, DMTF, http://dmtf.org/standards/cloud

RESERVOIR, Resources and Services Virtualization without Barriers, http://www.reservoir-fp7.eu

Rodero-Merino, L., Vaquero, L. M., Gil, V., Galan, F., Fontan, J., Montero, R. S., Llorente, I. M., (2010), *From infrastructure delivery to service management in clouds*. Future Generation Comp. Syst. 26(8): 1226-1240, 2010.

Tclouds, Tclouds Project, http://www.tclouds-project.eu/.

Youseff, L., Burtico, M., Da Silva, D., (2008), *Toward a Unified Ontology of Cloud Computing*, Grid Computing Environments Workshop.

KEY TERMS AND DEFINITIONS

Elasticity: the growing and shrinking of resource usage at run-time.

Federated cloud: a cloud that is made up more than one participant cloud. Each cloud is independent, but federates with other clouds to form a larger environment.

Infrastructure provider: an organization that provides the infrastructure, (i.e servers and networks), that hosts and executes the virtual machines in a cloud. The infrastructure elements are leased out to service providers.

Service cloud: a cloud that executes services on behalf of a service provider. This is different from a storage cloud or a network cloud.

Service Definition Manifest: The service provider defines the details and requirements of the application in a Manifest. This is done by specifying which virtual machine images are required to run.

Service provider: an organization that creates an application and deploys it as a set of virtual machines on an infrastructure providers facilities. When the application executes, it will be a service for its end-users.

SM: The Service Manager is the component responsible for accepting the Service Definition Manifest and the raw VEE images from the service provider.

VAN: a Virtual Area Network, implemented as virtual Ethernet overlay services, connects services running in virtual machines with one or more private virtual networks in order to keep them independent of other services.

VEE: a virtual execution environment, which can be a virtual machine, executes the code of a service. A service cloud operates by acting as a platform for running virtualized applications in VEEs, which have been deployed on behalf of a service provider.

VEEH: a Virtual Execution Environment Host is a resource that can host a certain type of VEEs. For example one type of a VEEH can be a physical machine with the Xen hypervisor controlling it, whereas another type can be a machine with the KVM hypervisor.

VEEM: the Virtual Execution Environment Manager is the component responsible for the placement of VEEs into VEE hosts (VEEHs) and is responsible for optimization of a whole site.