CHAPTER 2

Cloud Architectures and Management Approaches

Dapeng Dong, Huanhuan Xiong, Gabriel G. Castañe, and John P. Morrison

Abstract An overview of the traditional three-layer cloud architecture is presented as background for motivating the transition to clouds containing heterogeneous resources. Whereas this transition adds many important features to the cloud, including improved service delivery and reduced energy consumption, it also results in a number of challenges associated with the efficient management of these new and diverse resources. The CloudLightning architecture is proposed as a candidate for addressing this emerging complexity, and a description of its components and their relationships is given.

Keywords Cloud architecture • Infrastructure • Management • Service delivery model • Heterogeneous cloud

D. Dong (✉) • H. Xiong • G. G. Castañe • J. P. Morrison
Department of Computer Science, University College Cork, Cork, Ireland
e-mail: d.dong@cs.ucc.ie; h.xiong@cs.ucc.ie; gabriel.gonzalezcastane@ucc.ie; j.morrison@cs.ucc.ie

© The Author(s) 2018

T. Lynn et al. (eds.), *Heterogeneity, High Performance Computing, Self-Organization and the Cloud*, Palgrave Studies in Digital Business & Enabling Technologies,
https://doi.org/10.1007/978-3-319-76038-4_2

2.1 Introduction

Cloud end-users are demanding greater performance and diversity of cloud services than ever before. As discussed in Chap. 1, the high-performance computing (HPC) and other end-user communities are seeking to exploit new and diverse hardware designed for specialist tasks. As well as supporting these new demands, cloud service providers (CSPs) face the challenges of achieving cost-effective scalability while increasing energy efficiency. Accommodating heterogeneity and maximising server utilisation (and by inference minimising over-provisioning) is a significant shift from conventional homogeneous cloud computing service design. This is particularly the case with HPC where end-users require a greater level of access and control over elements of the cloud infrastructure. To access heterogeneous resources, exploit these resources to reduce application development effort, make optimisation easier, and simplify service deployment, a re-evaluation of our approach to both resource management and service delivery is required.

The remainder of this chapter discusses conventional cloud architecture designs and provides an overview of the CloudLightning architecture, a novel architecture designed to meet the challenges of the heterogeneous cloud. The next section presents the three layers of conventional cloud architectures—the Infrastructure Layer, the Cloud Management Layer, and the Service Delivery Layer. This is followed by a discussion of the main challenges associated with transitioning to a truly heterogeneous cloud with an emphasis on resource management and abstraction. In Sect. 2.4 CloudLightning is presented, a cloud architecture inspired by the design principles of emergence, self-organisation, self-management, and the separation of concerns discussed in Chap. 1. Each functional component and their relationships are detailed to provide insights into how it differs from the conventional cloud and realises important properties from the end-user and CSP perspectives including support for heterogeneity, ease of use, auto-scaling, data locality, high availability (HA), and networking organisation.

2.2 Cloud Architecture

Over the last decade, large-scale consumer-facing cloud services have been created by service providers such as Amazon, Microsoft, Google, and Rackspace. These data centres are large industrial facilities containing the computing infrastructure that runs their services: servers, storage arrays, and networking equipment. This core equipment requires supporting infrastructure in the form of power, cooling, and external networking links. Reliable service delivery depends on the holistic management of all of this infrastructure as a single integrated entity. Architecturally, this holistic management can be logically separated into three layers from bottom to top including an Infrastructure Layer, a Cloud Management Layer, and a Service Delivery Layer, as shown in Fig. 2.1.

2.2.1 Infrastructure Organisation

Cloud infrastructure design is the art of balancing requirements to ensure data centre scalability, maintaining server fault tolerance, minimising costs, and maximising bisection end-to-end bandwidth (Kim 2011; Wang et al. 2014). Traditional data centre infrastructure is based on a hierarchical structure typically with a three-tier design including the Access Layer, the

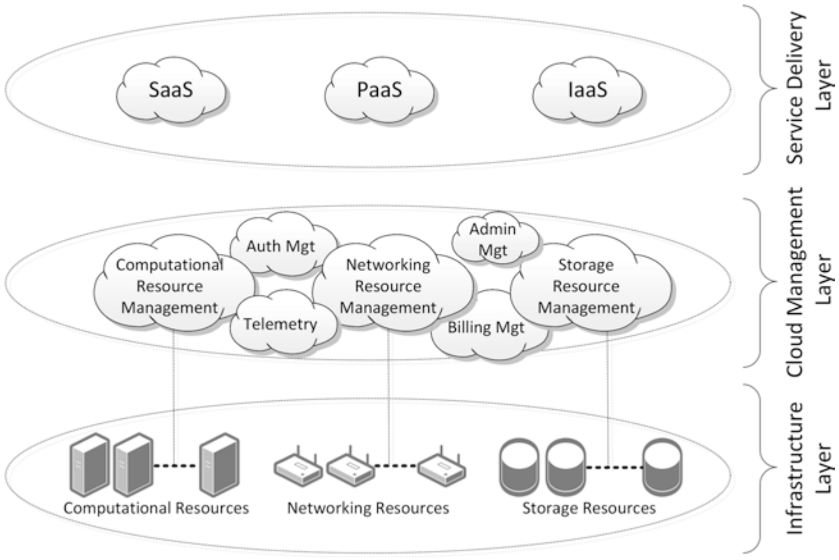

Fig. 2.1 Classical cloud architecture is considered to be composed of three layers. The Service Delivery Layer is one seen by users; this layer is realised by the Cloud Management Layer, which is also responsible for realising the objectives of the Cloud Service; the Infrastructure Layer comprises of the underlying collection of storage, computing, and network resources and their required hardware and software

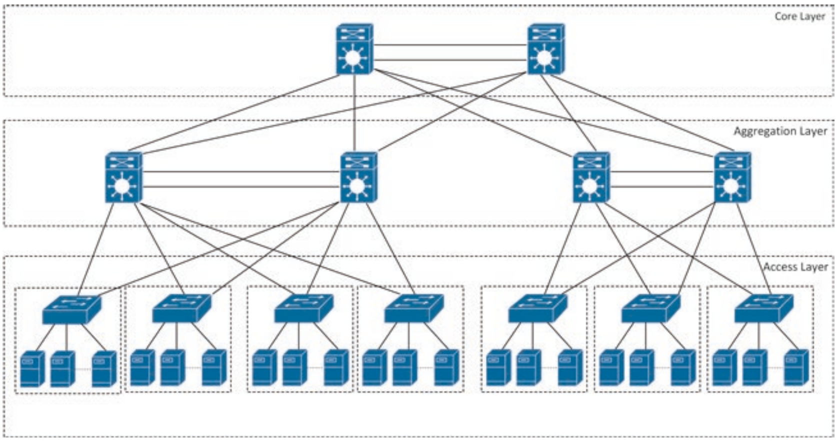

Fig. 2.2 The traditional three-tier networking infrastructure

Aggregation Layer, and the Core Layer (Martin Pueblas 2010), as shown in Fig. 2.2.

- *The Access Layer* (also called the Edge Layer): The primary function of the Access Layer is to connect servers that typically reside in the same rack. An Access-Layer switch is thus often referred to as a Top-of-Rack (ToR) switch.
- *The Aggregation Layer* (also called, the Distribution Layer): The Aggregation Layer is a multi-purpose system that interfaces the Access and Core Layers. The main function of the Aggregation Layer is to keep the various communication domains separately, thus providing intelligent switching and HA between regional ToRs.
- *The Core Layer*: The Core Layer is responsible for providing high-speed, scalable, and reliable connectivity across the entire data centre.

This traditional three-tier data centre design is created with simplicity in mind. The design relies on the use of high-end enterprise-class switches in the upper layers, whereas the lower layers can function effectively with less sophisticated equipment. Previous research has indicated that adding additional servers to a data centre, using the traditional three-tier design, will reduce the end-to-end bisection bandwidth in proportion to the size of the data centre (Al-Fares et al. 2008). In support of cloud computing and in response to the rise in popularity of Big Data and High-Performance Computing as a Service (BDaaS and HPCaaS, respectively), the organisation of the infrastructure in modern data centres is biased towards scalability and high throughput.

In general, design strategies are centred on two basic models—the Switch-Centric model and Server-Centric model. The next section discusses these models and the main network designs associated with these models.

2.2.1.1 The Switch-Centric Model

In the Switch-Centric model, servers are interconnected using switches and routers. The Fat-tree network is a representative of the Switch-Centric model that is widely acknowledged and used for data centre networking infrastructure. A Fat-tree network is also known as Clos topology (Leiserson 1985). In a Fat-tree network, servers are grouped into Points of Delivery (PoDs). A PoD consists of n number of servers and n number of switches. $n/2$ switches are connected to n servers and act as Access-Layer switches. The remaining switches are connected to the Access-Layer switches and, to each other, acting as Aggregation-Layer switches. Moreover, PoDs are connected using additional $(n/2)^2$ switches acting as Core-Level interconnections. Thus, the Fat-tree design guarantees a one-to-one over-subscription ratio between any pair of nodes in the network. However, the scalability of the infrastructure is limited by the number of ports available on each switch. BCube (Guo et al. 2009) is another Switch-Centric design based on a recursive-defined topology. In a BCube design, n servers are connected to an n-port switch forming a cell. n cells are connected through n switches to form a cube. BCube is designed for modular data centres and accommodates high performance in a multicast and broadcast network; however, the complexity of network cabling is relatively high. Portland (Niranjan Mysore et al. 2009), RBridges (Ghanwani 2011), SmartBridge (Rodeheffer 2000), SEATTLE (Kim 2011), and VL2 (Greenberg et al. 2011) are commonly used routing and forwarding protocols and network address schemes for the Fat-tree-based infrastructure.

2.2.1.2 The Server-Centric Model

In the Server-Centric model, both servers and switches participate in packet routing, and in the Server-Centric model, both servers and switches participate in packet routing and forwarding. DCell (Guo et al. 2008) is a representative implementation of the Server-Centric model. In DCell, n servers are connected to an n-port switch forming the smallest entity known as a Cell. $n+1$ number of Cells are interconnected via the network interfaces of each server, thus forming a larger network. The hierarchical topological design makes DCell networks scalable and robust. However, the network diameter increases exponentially with the size of the network. This implies that Cells in the inner layer will carry more network traffic, and end-to-end communications may experience greater latency. FlatNet (Lin et al. 2012) is another Server-Centric recursive-defined network. The FlatNet design uses more switches to achieve higher scalability, n^3, compared to n^2 of DCell. Based on similar rules used in DCell, FlatNet organises n servers in an n-port switch as a Cell. A higher layer is formed from n^2 number of lower layers. In FiConn configurations, the main network interfaces of a server are connected to their corresponding ToR switch(es), and the redundant network interfaces of a server is used to establish direct server-to-server connections (Li et al. 2009). In contrast to DCell, FiConn, and FlatNet, the SprintNet design focuses on high performance. SprintNet uses multiple, c number of switches connecting n servers in each Cell, in which $n/(c+1)$ ports connect to other Cells in the network. Infrastructure expansions are achieved by adding $c*n/(c+1)$ Cells each time. The SprintNet is specially designed for high-throughput infrastructure.

The current trend is towards using a Server-Centric design based on a recursively defined topology. From a cloud management perspective, the number of servers determines scalability, the number of switches affects the infrastructure cost and the energy efficiency, the number of links indicates the complexity of constructing the network, and the diameter of the network directly influences the network throughput (high-throughput networks will improve the service delivery experience, especially for Big Data and HPC and high-throughput computing (HTC) applications). HPC and HTC based on heterogeneous computational resources may have specific requirements on the types of switches, port numbers, and link capacity. Unfortunately, none of the existing design schemes can guarantee scalability, fault tolerance, high performance, and energy efficiency at the same time. To this end, a hybrid infrastructure organisation scheme using the combination of several interconnected topological designs may be required. For example, a combination of Fat-tree, BCube, and SprintNet may be capable of providing the required infrastructure. As a side effect, a hybrid design introduces further complexity that must be managed.

2.2.2 The Cloud Management Layer

Depending on the business goals, the technologies chosen to implement a cloud architecture varies from vendor to vendor. In principle, all cloud architecture implementations aim to realise quality attributes that most appropriately reflect the business goals of the CSP. In Chap. 1, cloud computing was defined, as per National Institute of Standards and Technology, as having five properties including on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service (Mell and Grance 2011). Technically, any data centre having those properties can be considered as a cloud. These properties can be realised by composing a set of commonly acknowledged functional components, as shown in Fig. 2.3. In principle, all cloud management platforms follow the same architectural design, but their implementations vary greatly. The following sections give a high-level overview of how two representative cloud management platforms, namely OpenStack and Google Kubernetes, implement the classical cloud architecture, based on virtualisation and containerisation technologies, respectively.

2.2.2.1 OpenStack

OpenStack (OpenStack, LLC 2017) is an open-source cloud platform designed to manage virtualised environments. Hypervisors are used to virtualise servers; various technologies including Virtual Local Area Networks,

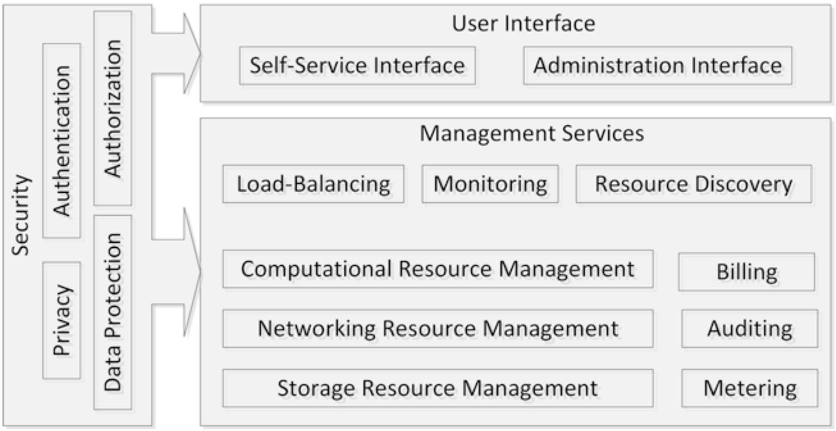

Fig. 2.3 Cloud management architect—a component view

Linux kernel namespaces, and various tunnelling techniques are used to virtualise networks; and storage resources are abstracted through the use of Network File Systems, Remote Volume, Object Storage, and other network-based clustering file systems such as GlusterFS (Red Hat & GlusterFS 2012), Ceph (Weil 2006), and Google File System (Ghemawat et al. 2003).

In particular, for managing computational resources, OpenStack uses a front-end Application Programming Interface (API) server for receiving and answering requests. Typically, allocating a computational resource will require other components, for example, a virtual network, a security group, and operating system images. This can be a complex task when dealing with multiple simultaneous requests with different configurations. In order to reduce this complexity, the front-end API server forwards the requests to a *nova-conductor* service. The *nova-conductor* coordinates various associated components to satisfy for a particular request. The *nova-conductor* uses a scheduler service (*nova-scheduler*) to locate potential physical server(s) that meet the specified requirements, including the number of Central Processing Unit (CPU) cores, the size of memory, and storage space. The requested resources (Virtual Machines [VMs]) will be deployed by a *nova-compute* service (by calling hypervisor-specific APIs) on the most appropriate physical servers. Architecturally, the computational resource management consists of a front-end API server, request coordinators (can be a group of resource coordinators to deal with high-volume requests), and an agent per computational node (executing the actual resource provisioning and deployment commands).

Managing networking in the cloud is a complex task. This is because conventional network functional components, for example, firewalls, routers, switches, networking connections, and Network Interface Cards (NICs), must be provided to end-users on top of shared physical networking resources and networking equipment. These cannot be virtualised or containerised like computational resources using hypervisors or container engines; rather, networking virtualisation is mainly built on top of several packet tagging/encapsulation techniques and the use of software implementations of respective networking devices such as virtual routers and virtual switches.

OpenStack storage systems are decoupled from computational resources. OpenStack offers several basic types of storage systems including traditional database systems, network-attached storage, and object storage. The back-end technologies supporting these storage systems vary greatly. In general, database systems and object storage are used by cloud applications, whereas remote volumes are used when creating VMs.

2.2.2.2 Google Kubernetes

Kubernetes is the most recent evolution of Google data centre management technology (Rensin 2015; Burns et al. 2016). Architecturally, Kubernetes uses a master/worker model. It consists of a master server and multiple minions (workers). The command line tools connect to the API endpoint in the master, which manages and orchestrates all minions. The minions receive instructions from the master and initialise local containers, appropriately.

A Kubernetes Master is composed of a number of components: the API server, the Replication Controller, the *etcd* Daemon, and the Scheduler. The API server is responsible for processing requests and for manipulating the underlying state objects. The Replication Controller determines how many pods or containers need to be run. The *etcd* Daemon stores configuration data. Lastly, the Scheduler is used to place work on an appropriate minion (or minions) based on an analysis of the state of the current infrastructure and the requirements of the service being provisioned.

A Kubernetes Minion is also composed of a number of components: the Kubelet, the Proxy, the cAdvisor, and a Pod. The Kubelet manages the lifecycle of containers in response to instructions from the master. The Proxy forwards network traffic to the appropriate containers. It performs primitive load balancing and is responsible for making sure that each networking environment is internally accessible while remaining isolated from other environments. The cAdvisor is a daemon that provides container users with an understanding of the resource usage and the performance characteristics of their containers. Finally, a Pod defines a collection of containers, deployed on the same minion, and provides them with a shared context.

2.2.3 The Service Delivery Layer

As outlined in Chap. 1, there are three basic cloud service delivery models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). These service delivery models are also referred to as cloud business models or resource abstraction models. Each of these delivery models is realised in specific layers of the cloud architecture. IaaS, for example, provides end-users access to tangible physical infrastructures, such as physical servers, networking equipment, and storage systems. IaaS also provides access to virtualised physical servers, known as Virtual Machines. IaaS offers maximum flexibility to end-users for configuring and operating the acquired resources, thus IaaS targets end-user groups interested in building Information Technology (IT) infrastructure.

In order to reduce the configuration complexity and operational costs, CSPs can provide pre-configured platforms and offer those ready-to-use platforms to the end-user. This service model is often referred to as PaaS. Examples of PaaS are pre-configured operating systems (e.g., Linux, Windows), Web application servers (e.g., Apache Tomcat, Oracle Glassfish Red Hat JBoss), Workflow Engines (e.g., Apache Orchestration Director Engine), and Messaging frameworks (e.g., RabbitMQ, ZeroMQ). PaaS provides services to system administrators and developers in need of pre-configured platforms for their systems or applications to function as expected. Although PaaS can greatly reduce configuration complexity and operational costs, it still requires the end-users to have domain-specific knowledge to engage with the platforms being provided. There are also cloud end-users who are interested only in consuming services, such as email, business processes, customised applications, for example, Customer Relationship Management and Enterprise Resource Planning. When a CSP has installed, configured, and provided those customer-facing software solutions as a service, they are referred to as SaaS.

As the cloud ecosystem rapidly evolves, heterogeneous resources are being incorporated into the cloud environment, which has traditionally been homogeneous. This evolution requires multiple service abstraction modes to coexist and to be combined to provide more versatile services.

2.3 Transitioning to Heterogeneous Clouds

Cloud infrastructure has traditionally been built on homogeneous resources. This approach afforded simplicity of design and uniformity of resource management. In recent years, different types of resources have been made available to the cloud user community and have proven to be extremely popular due to their speed and modest power consumption. This evolution on the tradition design is thus leading to the emergence of the heterogeneous cloud. Heterogeneity is a broad concept. It can refer to different models of physical servers, produced by various manufacturers, and/or it can refer to different servers having different computational power, storage size, and networking capacities. Functionally, various types of coprocessors and accelerators, such as the Intel Xeon Phi Coprocessor

(Many Integrated Core [MIC]), the Field-Programmable Gate Array (FPGA), and the Graphical Processing Unit (GPU), have already been used in many production clouds. At a lower level, each type of CPU (Advanced Micro Devices, Intel, or even Advanced Reduced Instruction Set Computing Machine [ARM]), system memory (e.g., Double Data Rate {1, 2, 3}, 3D transistors), and storage types (e.g., mechanical disks and Solid State Disks) has different speeds and power consumption patterns. From a networking perspective, several types of networking connections (e.g., 1 Gb/s standard Ethernet, 10/40Gb/s high-speed Ethernet, Fibre Optical network, and InfiniBand) coexist in many major cloud deployments. The heterogeneity in hardware, resource organisation schemes, and software creates rich features and services that can support a wide range of applications from general web applications and networking infrastructure services to Big Data processing, high-performance/throughput computation applications, and recently the Network Virtual Function to support traditional telecommunication applications.

Heterogeneity also has its challenges from a cloud management perspective due to the complexity associated with managing diversity. Each type of hardware, resource organisation scheme, and software has its own unique static features, such as architecture, computation power, speed, and bandwidth, and each also exhibits different runtime patterns, such as power consumption, computation performance, access methods, and supporting software libraries. In order to efficiently and effectively manage such complex environments, the Cloud Management Layer must adapt to this evolving diversity. In this regard, the two most challenging aspects that must be addressed are the efficient management of resources and the support for flexible resource abstraction methods.

2.3.1 Resource Management

Heterogeneous resources introduce a large feature space into the cloud. The careful refinement of resource features and their combinations provide two clear advantages: (i) support for a wide range of applications and (ii) an appropriate mapping between application requirements/specifications and the resource features/characteristics. These can maximise the desires of both the end-user and the CSP, for example, respectively maximising application performance and reducing power consumption. This process requires resource management capable of efficiently and effectively manipulating such a large feature space at scale.

In the current cloud environment, resource scheduling can be categorised into three schemes including Monolithic, Two-Level Scheduling, and Shared-State (Schwarzkopf et al. 2013).

A Monolithic Scheduler has a single instance, is sequential, and must implement all policy choices in a single code base. The Google Borg scheduler is effectively monolithic, although the more recent releases of this scheduler have been optimised to provide internal parallelism and multi-threading to address HA and scalability. A Two-Level Scheduling approach separates application schedulers from resource schedulers. Mesos acts in this manner. It is an infrastructure management framework and makes use of a central master scheduler to decide how many resources from the available pool can be assigned to a framework. An application scheduler, within each framework, then allocates resources to applications within its own domain. Finally, a Shared-State scheme uses a Shared-State Scheduling approach, supporting multiple parallel schedulers. Each scheduler is given a private, local, frequently updated copy of the global state for use in making local scheduling decisions. Once a scheduler makes a placement decision, it updates the shared copy of the global state in an atomic commit, and the time from state synchronisation to the commit attempt is called a transaction. Google Omega (Schwarzkopf et al. 2013; Burns et al. 2016) uses the Shared-State scheme. Omega schedulers operate in parallel using lock-free optimistic concurrency control. Omega is also designed to support multiple distinct workloads having their own application-specific interfaces, state machines, and scheduling policies.

Common cloud resource scheduling algorithms map applications to resources using resource availability metrics such as the number of available CPU cores, the free memory, the available storage space, and other system-state information. These schedulers use as little information as possible to make reasonable decisions in a timely manner. This approach is sufficient for a cloud composed of homogeneous resources. In contrast, heterogeneous clouds introduce a much higher degree of complexity for which conventional approaches to resource management are inadequate. Thus, new and innovative solutions are required to efficiently support the transition from the homogeneous to heterogeneous cloud.

2.3.2 Resource Abstraction

Current cloud management platforms are typically designed to manage either virtualised or containerised environments. Considering that the traditional cloud consists of homogeneous resources based on general-purpose processing units (CPU architectures) and standard hardware components, virtualisation and containerisation technologies have demonstrated their ability, in many production environments, to abstract standard hardware resources.

However, heterogeneity creates new challenges to existing resource abstraction methods. Specifically, many computation accelerators, such as MICs and GPUs, cannot be simply virtualised nor containerised without specific configurations being done at both the hardware and software levels. In particular, different models and manufacturers of the same type of computation accelerators may require different configurations on the host server (e.g., setting CPU features in the Basic Input/Output System and motherboard configurations) and in the software (e.g., changing kernel versions, updating operating system drivers, and choosing the appropriate hypervisor). This poses the challenge of how to flexibly use various resource abstraction methods to access different types of resources seamlessly.

2.4 The CloudLightning Approach

The CloudLightning architecture has been constructed in an effort to address the challenges resulting from the transition to the emerging heterogeneous cloud. It recognises that the complexities associated with resource management due to this transition are nontrivial, and it proposes the use of self-organisation and self-management as a potential way forward. Thus, the architecture is composed of components and services with the necessary support for self-organisation and self-management. The CloudLightning architecture demonstrates how specialised hardware can be seamlessly integrated and the problems of centralised resource management at scale can be addressed, whilst recognising the inevitable added complexity resulting from supporting heterogeneity. Figure 2.4 shows the overview of the CloudLightning architecture, including the Service Delivery Layer, the Cloud Management Layer, and the Infrastructure Layer.

2.4.1 Infrastructure Organisation

The infrastructure organisation of CloudLightning is reminiscent of the Warehouse Scale Computer concept in which the infrastructure is composed

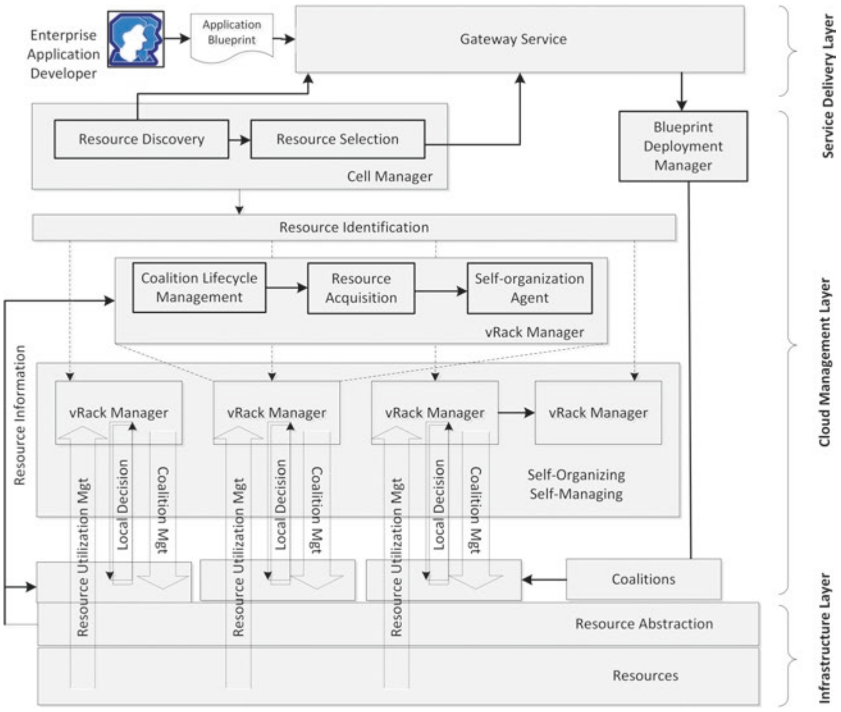

Fig. 2.4 An overview of the CloudLightning architecture showing how its various components are organised into the classical conceptual cloud layers

of *Cells*. A *Cell* is composed of Racks, which in turn contain servers of homogeneous hardware. In contrast, CloudLightning also incorporates heterogeneity by allowing different Racks to contain different computational resources.

2.4.2 *Hardware Organisation*

In a CloudLightning managed domain, physical servers are partitioned into groups based on geographical locations or regions; each of these partitions is called a *Cell*. A *Cell* is composed of a pool of heterogeneous computational resource, known as the Compute Resource Fabric. In the CloudLightning system, five elementary computational hardware types are considered explicitly. These include commodity servers (CPUs), servers with GPU accelerators, servers with MIC accelerators, servers with FPGA accelerators, and Non-uniform Memory Access Scale high-performance computer.

In a conventional data centre, physical racks are used to hold servers and switches. However, in a cloud deployment, the rack has no explicit identity that can be used to determine, from within the cloud software stack, where a particular compute/storage resource is physically located. To maintain information about groups of servers and to manage their resources, CloudLightning introduces virtual components called vRacks. A vRack contains a group of physical servers that share common properties including hardware type, hardware compatibility, and network connection type.

2.4.2.1 Resource Abstraction

The Hardware Abstraction Layer (HAL) provides a logical view of the underlying cloud infrastructure directly to the Cloud Management Layer. The HAL places resources into vRacks. Each vRack contains a certain number of homogeneous resources. The size of each vRack is initially determined by the management complexity for the type of resources to be managed. During the evolution of the system, a vRack may negotiate with other vRacks to exchange information and to transfer resources to achieve system goals such as maximising resource utilisation, reducing power consumption, and improving the service delivery experience.

When new hardware joins the CloudLightning managed domain, a dedicated Plug & Play interface is used to facilitate the connection of new hardware to the CloudLightning system. The newly connected hardware is required to expose available capacities and capabilities to the interface. In response, the interface will create CloudLightning-specific resources (CL-Resources) to represent the capabilities exposed. Depending on their type, these CL-Resources will be attached to an existing vRack, or if an appropriate vRack of this type is not available, a new vRack of an appropriate type is created. Where appropriate, the newly created vRack will be managed by a designated vRack Manager. This process is shown in Fig. 2.5.

2.4.3 The Cloud Management Layer

The CloudLightning management layer is shown in Fig. 2.4. The functional components and their relationships are explained in detail in the following sections.

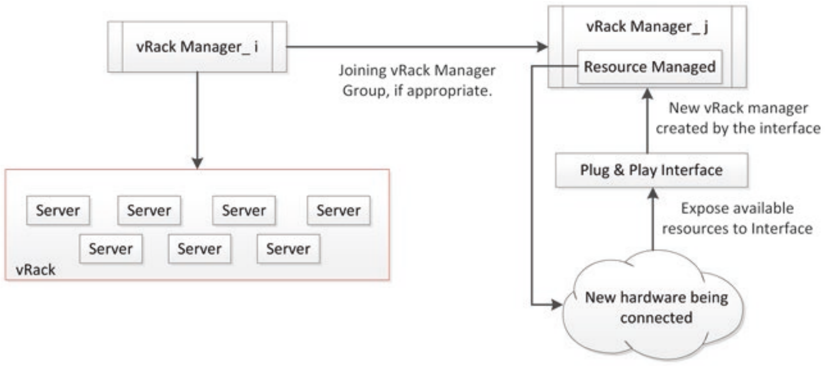
vRack Manager_ i
Joining vRack Manager Group, if appropriate.
vRack Manager_ j
Resource Managed
New vRack manager created by the interface
Plug & Play Interface
Expose available resources to Interface
New hardware being connected
Server
Server
Server
Server
Server
Server
Server
vRack

Fig. 2.5 Support for heterogeneous resources using Plug & Play interface at the Hardware Abstraction Layer

A Cell Manager is the software component associated with each Cell. The Cell Manager receives an *Application Requirements Document* from the Gateway Service, and it acquires CL-Resources in response to the requirements articulated in that "document". This can be done in at least one of two ways: either by allowing the user to select from a set of resources returned from a Resource Discovery phase or by allowing the system to assign appropriate resources immediately that meet the service requirements. In the former case, resource reservation is required while users make their choice, and in the latter case no reservation is needed.

2.4.3.1 CL-Resource Discovery

The CL-Resource Discovery process is initiated when the Cell Manager receives an *Application Requirements Document* from the Gateway. This "document" contains a set of Blueprint Requirements and a set of Service Requirements for each service in that Blueprint.

The function of the discovery process is to locate all of the possible CL-Resources that can be used to implement each of these services, consistent with particular constraints articulated in the list of Service Requirements.

The discovery process can determine information about dynamically changing capabilities and capacities by communicating with a group of vRack managers. From this information, the discovery process determines the CloudLightning system's ability to provide CL-Resources for each of the possible *Implementation Options* mentioned in the Service Requirements.

To guarantee these options remain available until the selection process is complete, all of the associated CL-Resources must be reserved by the associated vRack Managers. Thus, resources are potentially reserved across multiple vRack Managers until the selection process determines that they should be acquired or released. All of these *Implementation Options* are then passed directly to the CL-Resource selection process.

2.4.3.2 The CL-Resource Selection

This process applies the remaining constraints articulated in the list of *Service Requirements* and constraints associated with the *Blueprint Requirements* to determine a solution set consistent with all of the *Application Requirements*. If at this stage the solution set is not unique, the selection process will choose a unique solution by considering the options that minimise the overhead for the CSP. The associated CL-Resources in the solution set are then acquired automatically and those CL-Resources not in the solution set are released. Once the CL-Resources are acquired, their handlers are passed back to the Gateway for subsequent use by the Deployment Manager.

A vRack Manager is associated with each vRack. The function of a vRack Manager is to manage all of the CL-Resources that can be exposed from its associated vRack. In addition, it can create/aggregate CL-Resources in/on its vRack, as necessary. When the vRack Manager aggregates CL-Resources in its vRack, it creates a new type of CL-Resource called a *Coalition*. This is one of the defining characteristics of the CloudLightning system in that it allows CL-Resources to be formed into groups of homogeneous CL-Resource types to implement specific services with those requirements. A vRack Manager is responsible for managing the physical servers in its vRack. The set of servers associated with vRacks may be re-allocated over time. Similarly, new servers may be added to a Cell and others may be removed. This may trigger the creation/destruction/reorganisation of vRacks and their associated vRack Managers.

There are three functional components within each vRack Manager: a Resource Acquisition component, a Coalition Lifecycle Management component, and a Self-Organisation Agent.

2.4.3.3 Resource Acquisition

This component is activated by the selection process of the Cell Manager. It attempts to acquire CL-Resources; this can be guaranteed if they have been previously reserved. The CL-Resources being acquired may already exist within the vRack or they may have to be dynamically created by the vRack Manager. Once these CL-Resources have been acquired, their CL-Resource handlers are returned to the selection process of the Cell Manager.

2.4.3.4 Coalition Lifecycle Management

A Coalition is a special type of CL-Resource. In general, it represents a group of homogeneous CL-Resources, each of which exists within a single vRack. The vRack Manager may form a number of Coalitions, which may be persistent and used as a means of rapidly providing an implementation option for specific services. These persistent Coalitions are called *Static Coalitions*. The vRack Manager may also aggregate its CL-Resources, none of which may be a Coalition in itself, to form Coalitions dynamically in response to a specific CL-Resource acquisition request from Cell Manager. In managing dynamic CL-Resources, such as Coalitions, bin-packing strategies can be used to balance resource utilisation and power management. By appropriately managing the mix of static versus dynamic CL-Resources, faster service deployment can be balanced against potential savings on power consumption.

A Coalition is an entity that can be seen as an execution environment, formed by grouping together a number of CL-Resources. Coalitions may exist inside a single vRack and so each is under the control of single vRack Manager. The constituency of a Coalition may span multiple servers within that vRack. Coalitions are formed by a vRack Manager in response to specific service requirements. The vRack Manager may decide to persist Coalitions for improved service delivery, and these Coalitions are called Static Coalitions. Coalitions may also be formed dynamically by a vRack Manager again in response to specific service requirements. This dynamic formation may involve the dynamic creation of some or all of the constituent CL-Resources. When a dynamically formed Coalition is subsequently disbanded, its dynamically created constituents are destroyed, but any static CL-Resources used in its formation are left unchanged and persist to be reused in subsequent Coalition formations. Figure 2.6 illustrates a number of Coalitions in a vRack. From the illustration, it can be seen that a Coalition can exist entirely within a single server or can span multiple servers within the same vRack. In the situation that a single vRack Manager does not contain sufficient resources to satisfy a specific requirement, it may negotiate with an adjacent vRack Manager to acquire the appropriate resources.

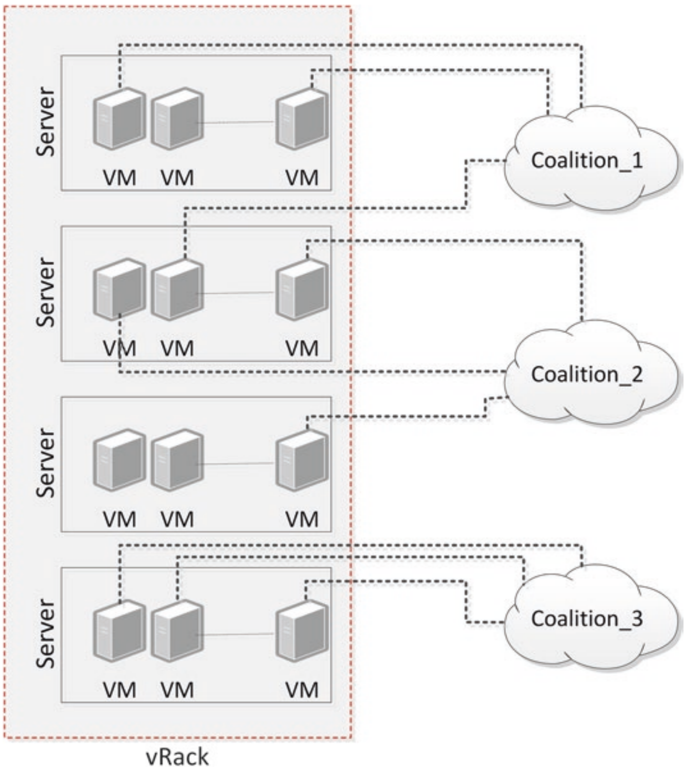

Fig. 2.6 Illustration of resource coalition

2.4.3.5 Self-Organisation Agent

The vRack Manager is a basic component of self-organisation in the CloudLightning system. vRack Managers organise themselves into groups and collectively determine local optimum strategies for CL-Resource management. The Self-Organisation Agent maintains information about other vRack Managers in the same group, it exchanges local state information with the Self-Organisation Agent in those vRack Managers, and it triggers power management decisions in the servers contained in its vRack. Negotiations between the various Self-Organising Agents within a vRack Manager group may result in the migration of servers from one vRack to another. Since CL-Resources may span multiple servers in the same vRack, any proposed migration must not violate the invariants associated with maintaining coalitions.

A vRack Manager Group is composed of a group of vRack Managers whose vRacks contain the same type of hardware. The Self-Organisation Agents of the vRack Managers within the group exchange information to optimally respond to resource discovery request from the Cell Manager. Together, they decide on if, and on where, the required CL-Resources are located or could be created. In making these decisions, the individual interests of each vRack Manager and the interests of the group as a whole are taken into account. This distributed decision process embodies the self-organisation strategy, which evolves to meet global objectives determined from the requirements driving the architecture design. vRack Managers are distinguished by the vRack hardware type. This distinction gives rise to a classification of the vRack Managers.

2.4.3.6 Classification of vRack Managers

Type-A vRack Managers are the most generic. They manage a collection of hardware resources of the same type (see Fig. 2.7). In one instance, these can be commodity hardware; in another, they could be CPU-GPU pairs, CPU-Data Flow Engine (DFE) pairs, or CPU-MIC pairs.

Type-B vRack Managers are more specialised. They manage a collection of HPC machines of the same type, each of which is exposed to the CloudLightning system as a single CL-Resource (see Fig. 2.8). If two or more HPC machines are managed by the same vRack Manager, then it is assumed that they are identical in all respects. This ensures that the CL-Resources exposed to the vRack Manager are the same.

Type-C vRack Managers manage a collection of hardware resources of the same type co-located on a high-speed interconnect (see Fig. 2.9). These can be commodity servers, or in other instances, they could be servers with GPU accelerators, servers with MIC accelerators, or servers with DFE accelerators.

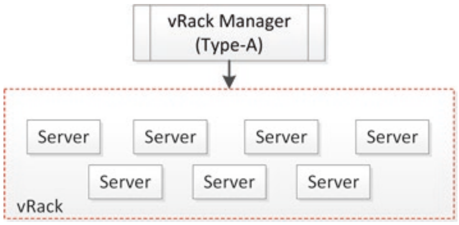

Fig. 2.7 vRack Manager Type-A

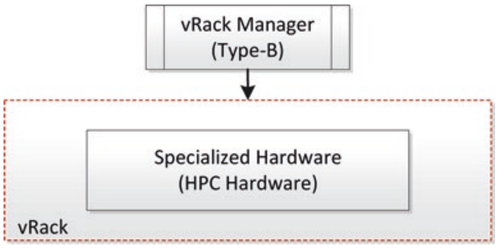

Fig. 2.8 vRack Manager Type-B

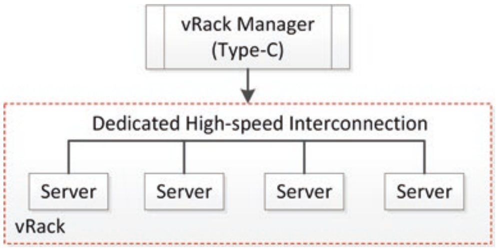

Fig. 2.9 vRack Manager Type-C

2.4.3.7 vRack Manager Activities

Type-A vRack Managers can only group with other Type-A managers (see Fig. 2.10). These groups can self-organise (e.g., in an attempt to improve power consumption). Self-organising involves servers migrating between vRack Managers in the same group. These groups also self-manage to improve service delivery but deciding locally which member of the group is the best to respond to particular service requests.

Neither Type-B nor Type-C vRack Managers engage in self-organisation. In general, the CL-Resources being managed are created from hardware of different types, thus cannot migrate to other vRack Managers. However, in principle, Type-B (see Fig. 2.11) vRack Managers can group together and Type-C (see Fig. 2.12) vRack Managers can group together in an effort to reduce the overall number of vRack Manager Groups. This in turn will simplify the administration required in the Cell Manager.

2.4.4 Service Delivery Model

The ready availability of large numbers of powerful, and increasingly heterogeneous, resources being made available by CSPs is making possible the deployment of large, data, and compute-intensive, applications. In many cases, these, quite often legacy, applications are monolithic in construction

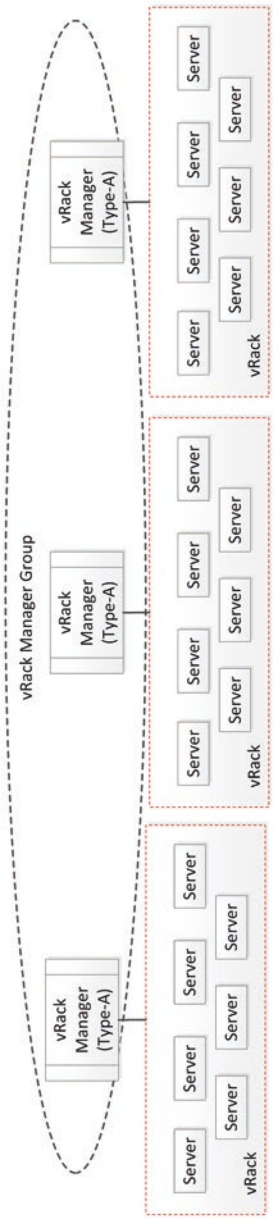

Fig. 2.10 vRack Manager Group Type-A

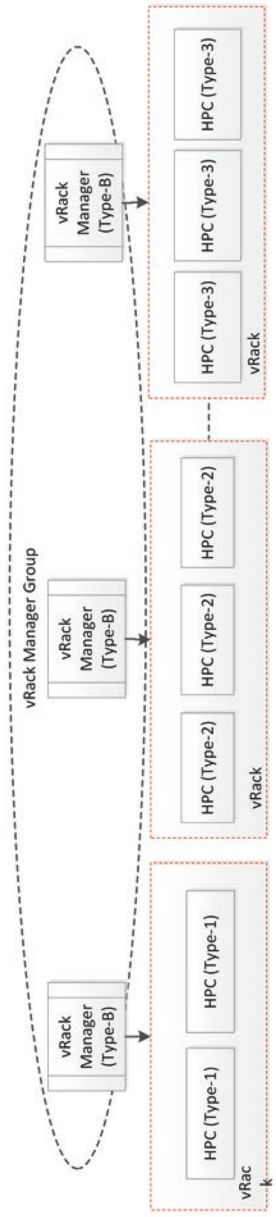

Fig. 2.11 vRack Manager Group Type-B

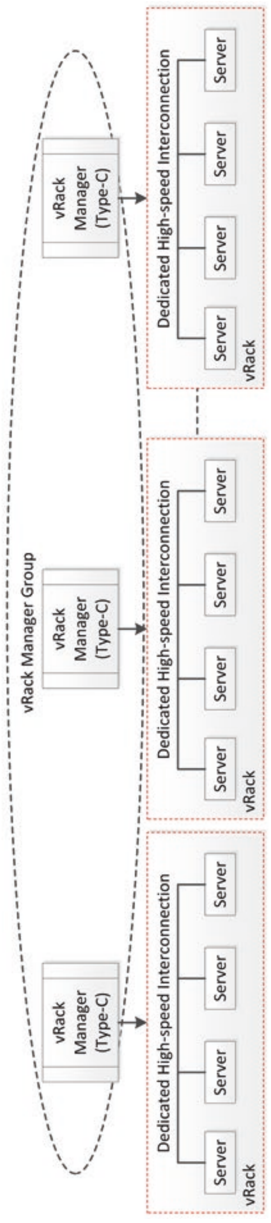

Fig. 2.12 vRack Manager Group Type-C

and require bespoke execution environments. Consequently, it can be challenging to deploy them in the cloud without acquiring IaaS and employing specialised engineering knowledge.

In this cloud usage model, the provider has no control over the effective utilisation of resources nor have cloud application developers an incentive to engage in costly customisation to increase resource efficiency when, regardless of the efficiency achieved, they are paying for the entire resource. Composing cloud services from workflows of large, possibly legacy, applications will most likely be the trend as support for emerging Big Data applications requires sophisticated, multi-phase data processing. Being essentially independent, the required resources for the applications that run in each of these phases may differ greatly in number and type, and hence the problems of cost and efficiency could be significantly exacerbated. Clearly, an approach is needed to allow the sophistication of the cloud to evolve in an efficient and cost-effective manner. It can be seen that there is no clear distinction between the concerns of cloud application developer and those of the Cloud Provider. The concerns of the CSP centre around efficient management and utilisation of cloud resources, and the concerns of cloud application developers centre on the specification, deployment, and service-level agreements (SLAs) associated with their applications.

To address this usability question, CloudLightning uses a Blueprint-oriented cloud application design and deployment approach. In this context, Blueprints are workflows in which nodes (*Service Element*) represent extant applications and edges distinguish the phases of the Blueprint execution where particular applications are active. All *Service Elements* are stored in a *Service Catalogue*, which is managed by the Gateway Service (Fig. 2.4). Cloud application developers may choose Service Elements from the *Service Catalogue* and link *Service Elements* to realise desired business logics. Attributes and parameters can be specified on a per *Service Element* basis. Altogether, the *Service Elements*, their linkages, and associated attributes and parameters comprise the application Blueprint, as shown in Fig. 2.13. The use of the Blueprint would drastically alter the current cloud usage model in that it would shift the burden of resource discovery, provisioning, and deployment from the cloud application developers to CSPs. This shift would greatly reduce the cost to, and the level of expertise needed by, cloud application developer while simultaneously giving CSPs full control over, and affording opportunities for the efficient use of, the cloud resources.

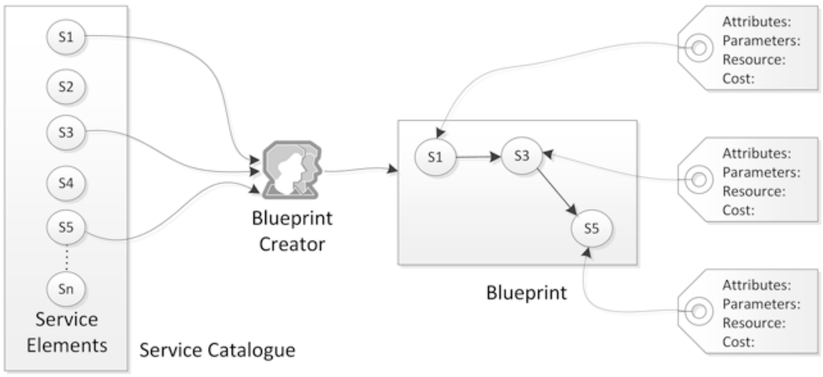

Fig. 2.13 CloudLightning Blueprint

2.4.5 *Advanced Architecture Support*

The design philosophy of the CloudLightning architecture is fundamentally different from the current cloud in operation. This results in the CloudLightning having different strategies for realising various important properties including auto-scaling, data locality, HA, and networking organisation.

2.4.5.1 Auto-Scaling

Scalability is one of the most important features in cloud computing. The CloudLightning system supports scalability provided that Blueprint developers explicitly indicate in the Blueprint which services are expected to require scaling. This explicit indication can be given by enclosing the services to be scaled within a *Scaling Envelope*. This envelope embeds services into Blueprint in order to monitor its load. When a pre-defined load threshold is crossed, this system service will dynamically acquire the appropriate resources from the CloudLightning system to scale the user service appropriately. By using the envelope in the Blueprint, consumers can see that execution of that Blueprint may result in charges relating to extra resources that cannot be determined statically. Additionally, the CloudLightning auto-scaling scheme allows application developers to explicitly specify how to service elasticity and partition data in a fine-grained manner. The scaling envelope and its associated impact on the CloudLightning system are illustrated in Fig. 2.14.

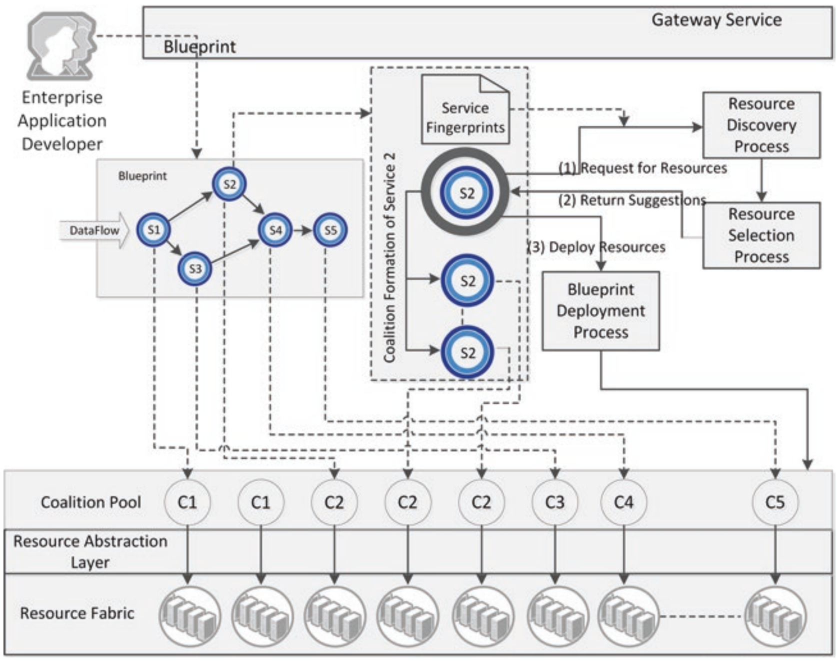
Gateway Service
Blueprint
Enterprise Application Developer
Service Fingerprints
Coalition Formation of Service 2
Resource Discovery Process
(1) Request for Resources
(2) Return Suggestions
Resource Selection Process
(3) Deploy Resources
Blueprint Deployment Process
Blueprint
DataFlow
S1
S2
S3
S4
S5
Coalition Pool
C1
C2
C3
C4
C5
Resource Abstraction Layer
Resource Fabric

Fig. 2.14 Auto-scaling using CL Envelope Mechanism

2.4.5.2 High Availability

HA refers to the mechanisms used to ensure continuity of service delivery. If an infrastructure component (e.g., network equipment or server) fails, redundancy and flexible load balancing mechanisms may be employed to ensure that the overall service remains available. HA will be addressed within the CloudLightning system by using a Hot-Standby server for each of its software components. To provide HA of the services running on the CloudLightning system, service replication may be used. Since replication has an associated cost, the decision to use it should be made by the Blueprint developers by expressing that preference in the Blueprint. An envelope mechanism similar to the one used for auto-scaling may be used.

2.4.5.3 Data Locality

Data locality, defined as keeping data close to the computation, is one of the most important factors considered for HPC/HTC and Big Data applications. In the cloud environment, the concept of data locality is not well defined. The CloudLightning system does not propose to introduce mechanisms to give Blueprint developers control over the data locality, unless that control is provided explicitly by specialised CL-Resources dedicated to high-speed data processing. Thus, this functionality would have to be exposed to the Blueprint developers at the Blueprint level.

In the CloudLightning system, data locality constraints may have to be considered at various levels in the self-managed and self-organised components; thus, it may be necessary to develop strategies for data locality at the Coalition, vRack, and Cell level. For instance, if a given Blueprint consists of two services: Service_A and Service_B, knowing that if Service_A will generate significant amount of data that will be further processed by Service_B (this information will be specified between Service_A/B in the Blueprint specification), then this information is a potential data locality requirement for the Blueprint which will be thereafter used by Cloud Management Layer to deploy the Blueprint on appropriate resources. On the other hand, in different application domains, such as HPC/HTC and Big Data, many applications require local storage for computation. In cases where data locality is a predominant concern, CloudLightning system is designed to use Network Attached Storages (NAS) through high bandwidth links in order to minimise the data transmission cost over the network. However, in cases where the NAS is not present, local persistent storage can also be used.

2.4.5.4 Dynamic VPN Creation for Blueprint Service Execution

To create an isolated execution environment for each Blueprint, the CloudLightning Management Layer creates dedicated Virtual Private Networks (VPNs) for each Blueprint, as shown in Fig. 2.15. The services within a Blueprint need to communicate with each other, services are mapped onto dedicated Coalitions, which may be running on different physical servers. In addition, the Coalitions running various services of a Blueprint may extend over multiple vRacks. Regardless of their physical location in the CloudLightning system, dedicated VPNs created for each Blueprint will ensure that CL-Resources and the data exchange between them remain secure and private to the Blueprint from which they are constructed.

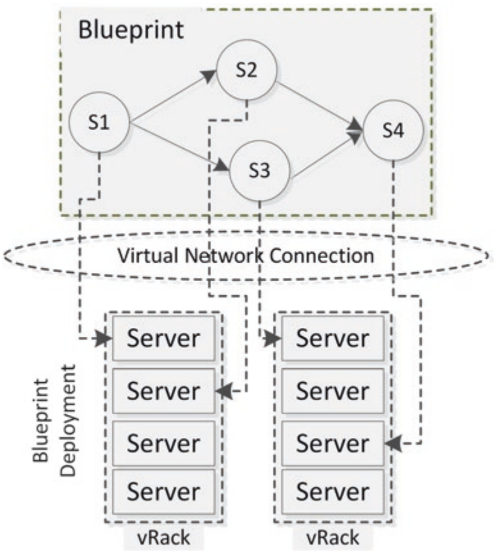

Fig. 2.15 Blueprint-driven VPN creation

2.5 Conclusion

The trend for hardware vendors to create more specialised offerings, capable of providing faster, more accurate, and power-efficient solutions, looks set to continue. The increasing demand for this hardware and for access to HPC is driving an evolution of cloud computing that offers versatile services. A heterogeneous cloud at scale embodies many hardware types, each with different cost/performance/power profiles. This, together with the attempt to satisfy the disparate needs of a large and varied customer community, makes the heterogeneous cloud a complex system. In evolving to heterogeneous clouds, CSPs may no longer offer Software/Platform/Infrastructure as a service, separately. Instead, CSPs may undertake to offer a combination of these to the customer on demand. This approach would require a service orchestration designer tool that could be used to compose a set of services together with an appropriate expression of service-level requirements into a cloud application Blueprint. From this perspective, customers no longer need to be concerned about how solutions are provided; rather customers can concentrate on describing the problem to be solved. This also gives more control to the CSP over how to provision and optimise resources, to meet both consumer needs and system requirements. However, the complexity of managing resources in a heterogeneous cloud environment should not be underestimated. Self-organisation is one of the tools that can be employed to effectively address this complexity. More specifically, in a properly designed self-organising approach, global system objectives may be met as the by-product of emergent behaviour resulting from the application of low-level self-organising rules and strategies; this approach has been adopted by the CloudLightning project. In the next chapter, the self-organising and self-managing approach for cloud management in the CloudLightning architecture level and details for developing effective cloud organisation strategies and efficient resource management algorithms are addressed.

2.6 Chapter 2 Related CloudLightning Readings

1. Xiong, H., Dong, D., Filelis-Papadopoulos, C., Castané, G. G., Lynn, T., Marinescu, D. C., et al. (2017). CloudLightning: A self-organized self-managed heterogeneous cloud. *Annals of Computer Science and Information Systems, 11*, 749–758.

References

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM Computer Communication Review, 38*(4), 63–74.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM, 59*(5), 50–57.

Ghanwani, R. P. (2011). *Routing Bridges (RBridges): Base protocol specification*. Internet Requests for Comments. RFC Editor.

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File system. *SIGOPS Operating System Review, 37*(5), 29–43.

Greenberg, A., et al. (2011). VL2: A scalable and flexible data center network. *Communications of the ACM, 54*(3), 95–104.

Guo, C., et al. (2008). Dcell: A scalable and fault-tolerant network structure for data centers. *SIGCOMM Computer Communication Review, 38*(4), 75–86.

Guo, C., et al. (2009). BCube: A high performance, server-centric network architecture for modular data centers. *SIGCOMM Computer Communication Review, 39*(4), 63–74.

Kim, C., et al. (2011). SEATTLE: A scalable ethernet architecture for large enterprises. *ACM Transactions on Computer Systems, 29*(1), 1.

Leiserson, C. (1985). Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers, C-34*, 892–901.

Li, D., Guo, C., Wu, H., Tan, K., & Zhang, Y. (2009). FiConn: Using backup port for server interconnection in data centers. In *INFOCOM 2009* (pp. 2276–2285). IEEE.

Lin, D., Liu, Y., Hamdi, M., & Muppala, J. (2012). FlatNet: Towards a flatter data center network. In *Proceedings of Global Communications Conference (GLOBECOM)* (pp. 2499–2504). IEEE.

Martin Pueblas, B. C. (2010). *Cisco service ready architecture for schools design guide*. Cisco Systems, Inc.

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing*. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

Niranjan Mysore, R., et al. (2009). PortLand: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Computer Communication Review, 39*(4), 39–50.

OpenStack, LLC. (2017). The openstack project. Retrieved from https://www.openstack.org

Red Hat & GlusterFS. (2012). *GlusterFS*. Retrieved from http://www.gluster.org

Rensin, D. K. (2015). *Kubernetes—Scheduling the future at Cloud Scale*. Sebastopol, CA: OSCON.

Rodeheffer, T. L. (2000). SmartBridge: A scalable bridge architecture. *SIGCOMM Computer Communication Review, 30*(4), 205–216.

Schwarzkopf, M. et al. (2013). Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (pp. 351–364). ACM.

Wang, T., Zhiyang, S., Yu, X., & Hamdi, M. (2014). Rethinking the data center networking: Architecture, network protocols, and resource sharing. *Access, IEEE, 2*, 1481–1496.

Weil, S. A. (2006). Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating Systems Design and Implementation* (pp. 307–320). USENIX Association.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (http://creativecommons.org/licenses/by-nc-nd/4.0/), which permits any noncommercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if you modified the licensed material. You do not have permission under this license to share adapted material derived from this book or parts of it.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.