Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Self-service infrastructure container for data intensive application

Ibrahim K Musa[1*], Stuart D Walker[1], Anne M Owen[2] and Andrew P Harrison[2]

Abstract

Cloud based scientific data management - storage, transfer, analysis, and inference extraction - is attracting interest. In this paper, we propose a next generation cloud deployment model suitable for data intensive applications. Our model is a flexible and self-service container-based infrastructure that delivers - network, computing, and storage resources together with the logic to dynamically manage the components in a holistic manner. We demonstrate the strength of our model with a bioinformatics application. Dynamic algorithms for resource provisioning and job allocation suitable for the chosen dataset are packaged and delivered in a privileged virtual machine as part of the container. We tested the model on our private internal experimental cloud that is built on low-cost commodity hardware. We demonstrate the capability of our model to create the required network and computing resources and allocate submitted jobs. The results obtained shows the benefits of increased automation in terms of both a significant improvement in the time to complete a data analysis and a reduction in the cost of analysis. The algorithms proposed reduced the cost of performing analysis by 50% at 15 *GB* of data analysis. The total time between submitting a job and writing the results after analysis also reduced by more than 1 *hr* at 15 *GB* of data analysis.

Keywords: Cloud computing; Microarray data analysis; Bioinformatics; Cells-As-A-Service

Introduction

Large scale data are increasingly generated from a wide variety of sources such as scientific experiments and monitoring devices. Consequently, there is a compelling need to store, analyse, query, manage, understand, and respond to such data for knowledge extraction and decision making. The emergence of cloud computing presents a new and promising paradigm to handle these challenges [1]. The model allows the use of configurable resources on a pay-as-you-go basis, thereby eliminating upfront investments. Scientific clouds [2] deployed on large heterogeneous research projects have shown good performance in handling burgeoning data volumes in an economic and efficient manner [3].

As the interest in cloud adoption for scientific applications intensifies, it is necessary to cope with the challenges of adhering to service level agreements, achieving high service elasticity, and the complexity of managing large scale cloud datacentre resources [4-6]. Another critical research issue [1,7] in cloud computing is the notion of enabling users to automatically consume cloud services without necessarily understanding the complexities associated with the new paradigm. To address these challenges, numerous techniques including effective provisioning strategies [8,9] and flexible job allocation [9] have been proposed.

This article presents a container-based model of cloud computing where all resources (virtual machines, storage, and interconnecting networks) and the logic to manage these resources are packaged in a virtual container and delivered to users. We refer to this model as Virtual Cells-As-A-Service (vCAAS) and each container as a vCell. The paper proposes a strategy similar to the research in [10,11]. vCAAS is an IAAS/PAAS model enabled with application specific resource management functionalities such as provisioning, job allocation, and holistic optimization [12]. The functionalities are created from platform as a service (PAAS). The resources are then consumed as an IAAS service similar to the Biolinux virtual instance.

Our proposal is a self-service container model inspired by the concept of a biological cell [13] on the premise that nature has successfully managed complexity. This

*Correspondence: ikmusa@essex.ac.uk
[1]School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, Essex, UK
Full list of author information is available at the end of the article

new approach aims to address the challenges of using cloud services for large scale data analysis. Our approach assumes VMs interact and complement each other to perform tasks. In vCAAS, each vCell is isolated from other tenants in the datacentre and the owner controls the entire components of the service cell from a simple template via a service console. The virtual container model provides a portable infrastructure platform for deploying self-contained and self-service IAAS.

This work demonstrates the strength of our model with an analysis from the area of bioinformatics. The GeneChips technology called GeneChips from the company Affymetrix is widely used by biologists and other life scientists to perform experiments on tissue samples. We use the microarray datasets from GeneChips in the analysis of RNA sequence to measure tissue samples. We use the microarray datasets to determine whether significant bias is encountered in this gene expression data. This bias could be introduced by the technology of the GeneChips rather than by the biology being tested. The work follows on from that pursued by Shanahan and colleagues [14].

The rest of the paper is organized as follows. The Section "Related works" presents related works in the area, the Section "Container-based cloud framework" describes our proposed framework, the Section "Container-based cloud model for bioinformatics" describes the specific implementation of the framework for bioinformatics experiment, the Section "Implementation and results" presents the experiment set-up and the result of experiment conducted, and finally the Section 'Conclusion' ends the paper and suggests ways in which the research can be expanded.

Motivation

Despite numerous developments in the adoption of cloud computing for scientific research, scientists employing the capabilities of cloud computing to perform experiments face the challenges of achieving efficient and cost effective use of cloud resources. This is attributed to the need to understand often complex cloud specific technologies (e.g. virtualization, job scheduling), to build and submit the required work flow, to choose the right virtual machine for the problem, to configure the purchased resources for the chosen experiment, and to choose the right number of resources for the given task [3,15]. Numerous projects, such as Biolinux [16,17], attempt to solve some of these problems by allowing cloud users to clone and use fully-packaged virtual machines containing scientific data analysis tools. This approach unifies scientific activities, reduces time to perform experiments, and offers cost-effective infrastructure. However, the challenges of efficiently and effectively running a large number of such packaged workstations still need to be addressed. As a result, this work is motivated by the following issues:

- Most current cloud computing models applied to solve problems in application areas (e.g. Biological sciences, Physics) are yet to clearly address the complete automation of domain specific resource management tasks such as initial capacity and the internal organization of resources as a holistic entity. Such intelligent resource management is needed to properly unlock the full advantage of cloud computing.
- Cloud users from specific domain areas are forced to understand and perform complex resource management tasks, thereby making the application of cloud computing in these areas more difficult.
- The need for cloud applications to explore the economic benefit of a holistic and complimentary interrelationship between set of purchased resources deployed for the execution of a task.
- The need to achieve more transparent interaction between a container management service and the underlying physical infrastructure.
- The need to implement proactive job monitoring services that apply historical pattern inference to determine status of jobs. Most current cloud platforms are yet to provide a full monitoring service at the job level. This often leads to high cost as a running VM with a failed job incurs unnecessary expense until the problem is detected and the failed job reassigned. This task is often left to the scientist running the experiment.

The main aim of this article is to present a next generation cloud computing model based on a holistic organization of resources in a virtual infrastructure container. The proposed framework optimises the provisioning of resources, automates the coordination of activities in a virtual service container, and efficiently allocates task to the resources provisioned as part of the container.

Related works

Enormous interest surrounds the application of cloud computing to support large scale biological data analysis [2,18-22]. A distributed system enabled by an intelligent agent in [19], data and software sharing using a central repository [18], and a publicly available packaged VM such as Biolinux [16] are already proposed. At extremely large scale, research projects such as CloudBurst [23], CloudBlast [24], and Galaxy [25] provide standard environments and algorithms for analyzing large data generated from scientific experiments. These development paves the way for cloud-based data analysis for bioinformatics. For instance, in [26] more than 1 billion short sequence reads were proposed using a cloud base algorithm available in [23].

Various models of holistic infrastructure services for cloud computing have been proposed [10,27-29]. Perhaps the proposals in [27,29] and [28] are the closest to the approach in this article. The proposals describe PAAS/IAAS container-based next generation cloud architecture created as a subset of a typical datacentre. The FP7 project 4Caast [29] described an enabling platform for an advanced PAAS cloud capable of offering an optimized programming interface for next generation internet application. The strategy outlined in [27] envisions the provision of compute, storage, and network services to a large number of multi-tenants each with specific performance criteria such as delay, security, and flexibility all defined in an Extensible Mark-up Language (XML) template. Each service user is assign a view isolated from other services. Another study in [28] describes the integration of both PAAS and IAAS in a VM set model. The proposal in [28] uses a data repository to store the configuration about available hardware and virtual resources which can be accessed using service oriented infrastructure (SOI). The attempt in [10] is to improve on the scalability of cloud services with a holistic resource view across a multi cloud environment.

Strategies for data transfer, cloud resource provisioning and job allocation have been proposed in the literature. A Stream-based data transfer strategy presented in [21] proposed a data compression technique to reduce the data transfer overhead associated with large data. In [30], various strategies for provisioning of virtual machines and workflow scheduling are described. Another similar survey in [9] tested various on-demand execution and waiting time provisioning policies.

vCAAS and other cloud computing models share many functional and structural features of existing technologies such as service grid, utility, and cluster computing (CCom). Cluster computing is a type of parallel and distributed system, which consists of a collection of interconnected stand-alone computers working together as a single integrated computing resource [31]. Computer clusters are often built around proprietary technologies and applications needs to be re-architected to meet policies [32]. In cluster computing, a service model is virtually absent and limited user requirement integration is offered during the resource composition. vCAAS, on the other hand, emphasizes user-driven service delivery of virtual components with the aim of creating business value. Service grids are, on the other hand, aimed as collaborative ventures with no apparent business objectives.

The VPC adopts a logical view where a cloud infrastructure appears as though it is operated solely for an organization. It may be managed by the organization or a third party and may exist on-premises or off-premises or some combination of these options [33]. Current deployment models of VPC concentrate mainly on extending an existing public cloud with secured virtual private network(VPN) services. On the other hand, the original concept of vCAAS, like similar previous proposals, envisions a flexible [27], user-driven [28], and self-managed [29] cloud service beyond the extension offered by laying VPN atop a public cloud as proposed in the VPC [7]. vCAAS aim to provide automated provisioning of resources, allocation of task, and failure management.

The next section describes our proposed framework for a virtual infrastructure container as well as the functionalities and actors coordinated to realise the vCAAS model.

Container-based cloud framework

The main component of our proposed framework (Figure 1) are the Delivery layer, Service layer, Control and Virtualization layer, and Physical fabric layer.

Delivery layer

This layer facilitates the submission of a vCell request and provides the enabling environment for the vCell. Technical requirements such as bandwidth, load balancing, priority, service discovery mechanisms, communication protocols, delay, and response time are properly identified and submitted using appropriate description language. Business objectives such as deadline, cost, accounting, and auditing are also submitted using this interface. Figure 1 shows a typical virtual infrastructure container framework.

Additionally, as shown in Figure 1 each vCell is enabled with the delivery layer which consist of four main subcomponents that interact to realize vCell functionalities.

The service console

enables the functionalities to interact with users and accept/update requests for vCell creation. The module allows simple and complex workflow specifications. VXDL [34] allows the complex request presentation required in a holistic cloud delivery model [28]. Complex parameters such as virtual timeline description, dynamic resource configuration, and components' behaviour during the service life cycle can be modelled. Fundamental features such as aggregate capacity and lease time of the submitted request is, however, subject to approval by the underlying service layer. This way the mediation function of the service layer controls and secures the subsequent activities of the vCell. For instance, a vCell manager cannot create resources beyond the maximum capacity or lease period.

Report interface

This provides monitoring capabilities to report (to the vCell Manager) jobs and virtual resource status and activities in the vCell. The module maintains and updates the

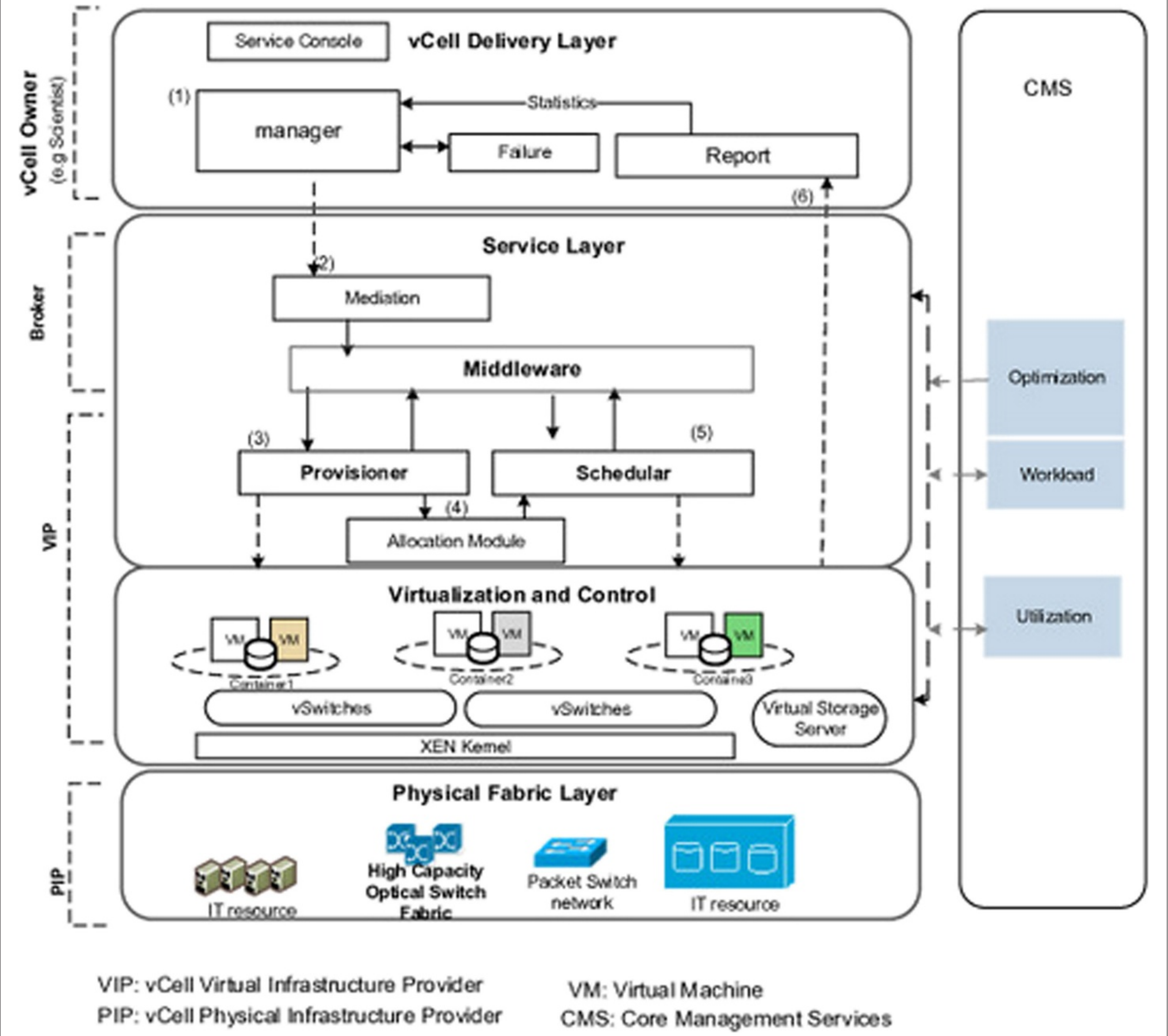

Figure 1 **Framework and interaction of various components in the proposed virtual infrastructure container.**

activities database of all components in the vCell. It maintains the record of finish times and provides time series data to the cell manager for use in predicting future task finish times. The functionalities provided by this interface enable additional failure detection and management capabilities. Each report is a tuple (S, T, F) of source, S, traffic type T, and feature F. The traffic type can be update, completion time, processed data e.t.c. F is a 2 tuple (Z, U) with size Z (e.g.in bytes or milliseconds) and unit U (e.g. s, MB, GB, Gbps).

vCell manager

All dynamic behaviour of the vCell is organized and coordinated by this module. This module is projected with privileged functionalities to perform management functions at cell level. This effectively reduces the management spaces making them small and correct. All ingress and egress traffic is controlled by this module. All the vCell managers are configured to send and receive messages in the cloud datacentre. However in receiving messages only those targeted to the vCell are treated while all others are discarded. This forms the initial basis for vCell isolation where all other components in the vCell are not exposed to receiving such messages - thereby isolating the components from broadcast messages. Although the manager receives these high communication signals, all other components are shielded from unnecessary communication overhead. The obvious problem is the performance of the CM. This research envisions cell-based management that eliminates high communication overload at the CM's interface.

Central to the functions provided by the module is the template-base mechanism for organizing and controlling the activities of components in a vCell. Each vCell is identified by a template which is used by the manager to create components, allocate resources, reconfigure existing components, and achieve all interactions with the underlying infrastructure and with other vCells. The request description received at the delivery layer is stored as a template for organizing the activities of vCell.

The request for vCell creation is submitted to the middleware residing in the global service layer. The service layer then creates the CM with basic service management functions projected into it as a privileged virtual machine. Subsequently, any additional component to be created in the vCell is initiated by the CM which queries available capacity and creates the required resources. The virtual switch connecting the newly created component (if computing or storage node) is added to the list v of virtual switches controlled by the CM. Two basic communications allowed in vCAAS are Intra and Inter vCell communication. In the former, two or more components interact to fulfil the submitted task. The vCell manager (CM) is equipped with the necessary control functionalities to configure network slice on the vSwitch. To isolate each of the vCells, a FlowVisor [35] is installed between the flows controller and cell manager. This way the CM control flows in the slice allocated for the vCell. This also provides to the vCell owner an illusion of unique isolated network infrastructure.

Inter-vCell communication exists between multiple vCells owned by one entity. This form of communication is regulated by the two vCell Managers in the communicating vCells. In the middle is the mediation layer which provide additional regulation by identifying the communication request and applying drop or allow policy. An example of the mediation layer is the SDN controller and a vSwitch. The controller maintains a set of rules each identified by the source and the destination of the traffic. The rules are matched and the appropriate decision such as forward to specific port, drop traffic, or add to certain queue is inserted into the forwarding table of the distributed virtual switch. Next time the same traffic appears, the vSwitch is equipped with the right set of rules in its forwarding table for the inter-vCell traffic.

Service layer

This enables the functionalities for container-based cloud service creation. It performs the provisioning of resources (e.g. CPU time, memory, storage, and network bandwidth) to a vCell, interacts with the underlying layer, and performs additional global scheduling. Interpretation of a submitted vCell request, and virtual resource requests are mediated by the layer. Privileged functionalities such as resource provisioning and job allocation are projected into each vCell during creation. The Template Engine provides functionalities for parsing job requests submitted by the owner (or function). The layer is capable of parsing both XML and Virtual Infrastructure Description Language (VXDL) file formats into canonical parameters.

Information about provider resources available for vCells to purchase are published as set of service units which can be acquired via the mediation layer. In Figure 2, WS-Net and WS-IT provide information about available network (e.g. bandwidth, links) and IT(computational and storage resources) respectively. The privileged VM in a vCell is capable of accessing the service units for scaling or task execution.

Virtualization and control layer

In the vCAAS model, an adaptive virtual interface (VIF) is created for each virtual node (VM). The VIF connects a VM to a distributed virtual switch (vSwitch) forming a virtual link. Combinations of these virtual links and connected virtual nodes (Figure 2) constitute the definition of a virtual network topology (VNT). This article employs an adaptive vSwitch-based traffic shaping strategy for vCAAS. Each VM in a vCell is assigned a virtual switch (vSwitch) port with initial bandwidth, based on request, and allowed to expand capacity relative to the unused bandwidth in the vCell. This way, the VMs adapt dynamically to various traffic conditions under the control of a distributed vSwitch (DVS) enabled with openflow protocol. To protect the service level agreement (SLA) of each container, traffic isolation mechanisms are applied at DVS level to control VM activity and one vCell does not have an adverse effect on the performance of other VMs in different vCells. The DVS maintains a database of entries for the policies in the network. Functionalities enabled by this module are under the control of a virtual infrastructure provider.

In the isolation strategy for vCAAS, each VM interacts with the vCell manager and other VMs, in hub and spoke arrangement, to complete work flow tasks. A VM experiencing high traffic (such as a storage VM) consumes high vCell's bandwidth. Other VMs require less bandwidth. At the time of creating a vCell, a variable super port with the equivalent capacity of bandwidth requested by a vCell is assigned. This super port can be viewed as an aggregation of fixed capacity ports. The bandwidth available for any VM is computed as:

$$A_{i,k} = B_{i,k} + \sum U_{j,k} \qquad (1)$$
$$i, j, k = 1, 2, \ldots N, \; i \neq j$$

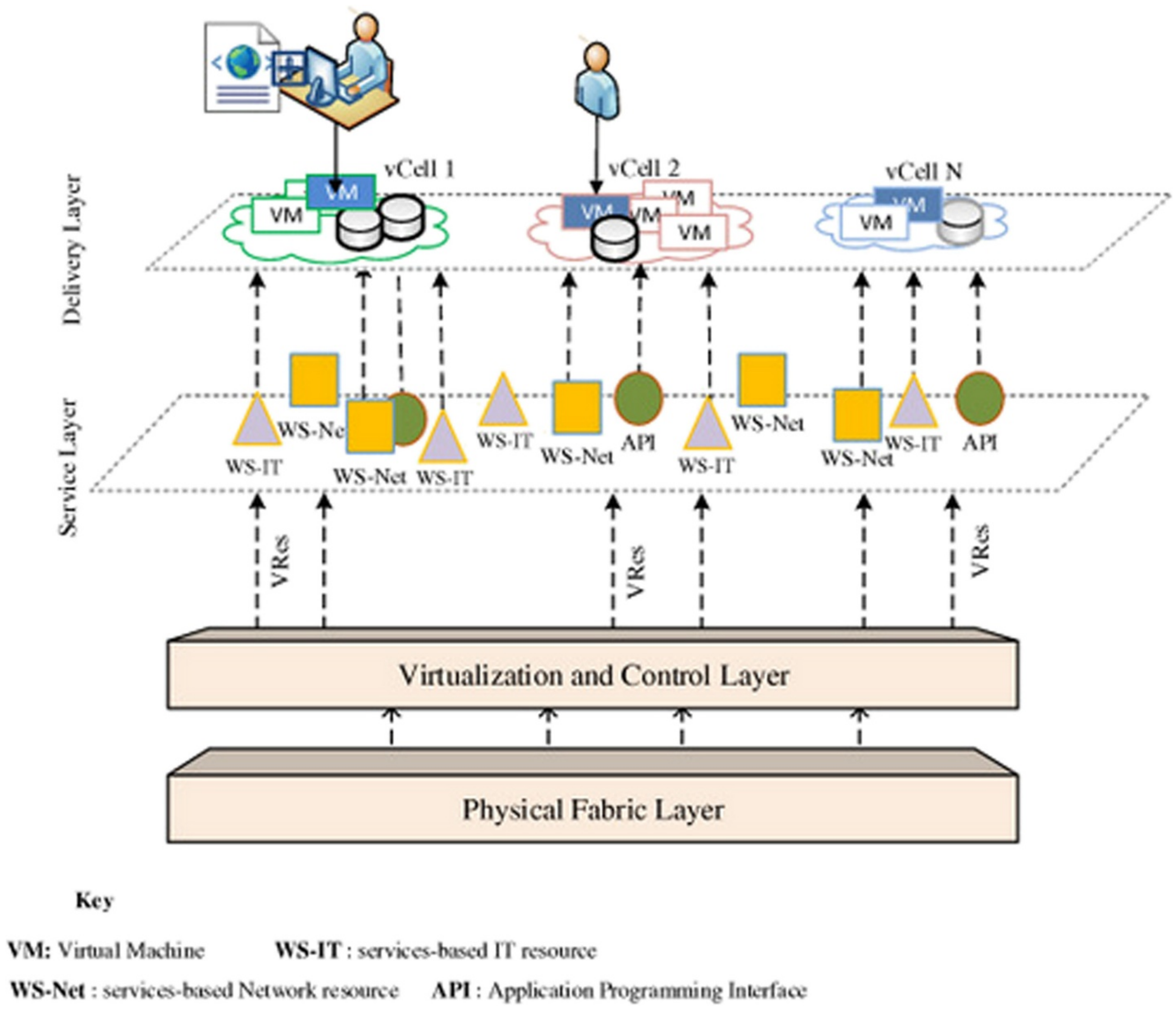
Delivery Layer
Service Layer
vCell 1
vCell 2
vCell N
VM
WS-Net
WS-IT
API
VRes
Virtualization and Control Layer
Physical Fabric Layer
Key
VM: Virtual Machine
WS-IT : services-based IT resource
WS-Net : services-based Network resource
API : Application Programming Interface

Figure 2 Service and delivery layers of the proposed framework (Resource view).

where Bi;k is the allocated bandwidth for VM i in virtual container k, Uj;k is the available bandwidth of VM j in virtual container k. The summation on the right of equation 1 is the unused bandwidth in vCell which can be utilized by a transmitting VM. Thus all bandwidth request β_i from VM i are accepted as long as $A_{i,k} \geq \beta_i$. This is demonstrated in Figure 3.

Physical fabric layer

This layer comprises heterogeneous physical IT interconnected with electrical and optical network in a hierarchical topology to enable cloud service. Network management services (NMS) are installed by the and access allowed to create a virtual resource.

Container-based cloud model for bioinformatics

This section presents the real world application of our virtual service cell. The concept of packaging application logic and resource management logic as a holistic resource container is implemented to perform large biological microarray data analysis. Virtual machines and the interconnecting network are created from a subset of the cloud datacentre and are configured to coexist and operate in concert during the execution of the data analysis tasks. Furthermore, the fundamental logic for job specific resource provisioning, jobs allocations, and optimization strategies [12] are packaged in a privileged virtual machine and delivered as part of the container. The framework for the cloud based container is shown in Figure 4. The section also presents the models adopted for the vCell automation.

Listing 1 Request for a vCell

```
<vxdl:vCell>
<vxdl:storageLocation>.6.1/NFSServer/GSE</vxdl:storageLocation>
<vxdl:vm>
        <vxdl:PE>2</vxdl:PE>
        <vxdl:MinMemory>2048</vxdl:MinMemory>
        <vxdl:NIC>1</vxdl:NIC>
        <vxdl:vNet>
             <vxdl:Duration>Read</vxdl:Duration>
              <vxdl:networkName>R</vxdl:networkName>
        </vxdl:vNet>
        <vxdl:vNet>
                  <vxdl:Duration>Write</vxdl:Duration>
                   <vxdl:networkName>W</vxdl:networkName>
             </vxdl:vNet>
        <vxdl:vNet>
                  <vxdl:Duration>Computation</vxdl:Duration>
                   <vxdl:networkName>C</vxdl:networkName>
        </vxdl:vNet>
        <vxdl:hdSize>
                <vxdl:min>20</vxdl:min>
                <vxdl:unit>GB</vxdl:unit>
        </vxdl:hdSize>
</vxdl:vm>
<vxdl:network>
        <vxdl:name>R</vxdl:name>
        <vxdl:Number>1</vxdl:Number>
        <vxdl:BW>
             <vxdl:min>2</vxdl:min>
             <vxdl:unit>GB</vxdl:unit>
        </vxdl:BW>
        <vxdl:dedicated>true</vxdl:dedicated>
          <vxdl:timeline>T1</vxdl:timeline>
</vxdl:network>
</vxdl:vCell>
```

Various roles

In the proposed model, clear separation of actors ensures flexibility of vCell creation and isolation. Various actors (Figure 5) interact to achieve the proposed vCAAS. Physical resources owned by vCAAS physical Infrastructure Providers (vCAAS-PIP) are accessed and virtualized by vCAAS Virtual Infrastructure Provider (vCAAS-VIP). vCell service starts with the submission of requests from the scientist performing the experiment - who also acts as the vCell owner (IAAS or PAAS user). The vCell owner submits a request for specific composition of network and computation to the Broker/vCAAS-VIP. Where a Broker exists, the actor then queries all vCAAS-VIPs and selects one. To meet the requirements of the vCell owner, the Broker evaluates the existing resource pool and consults vCAAS-VIP to appraise available resources suitable for the submitted request. vCAAS-VIP then virtualizes and offers the vCell to satisfy the initial request (Figure 5). The proposed framework combined the functionalities of the actors and is realised as an IAAS/PAAS model. The sequence of operation for the interaction between the various actors during the creation of a vCell are shown in Figure 6.

vCAAS automation

To properly operate as a self-service infrastructure for data intensive applications, each vCell is capable of the following key features:

1. Automatically compute the initial and dynamic capacity requirements of the container. There should be mechanisms for estimating the initial VMs and the configuration requirements of the data analysis to be performed by the vCell.
2. Optimally allocating the submitted jobs to VMs using the estimated current and future states of virtual machines in the vCell.
3. Adapt to the changing capacity availability in the virtual container in a holistic manner.

Enabling these features in vCell requires a dynamic model with the mechanism for continuous learning and update. This way, properties of the vCell and the resources contained can be properly characterized and managed without user intervention.

A Markov Chain offers a type of automation model that uses a stochastic process to determine or estimate the states of a system in a tractable manner. The Markov process uses observable and hidden features of the components to estimate future states. The latter feature is enabled by a kind of a hidden Markov Model (HMM) [36].

Within a vCell, component's status, such as the total workload, consists of both observable (e.g. average throughput) and hidden features (e.g. operating system's background processes). Using HMM in vCAAS enables automation features by estimating the future status of resources in vCell and reconfiguring components to adapt without user intervention. HMMs have been successfully applied in many problem domains including speech recognition, bioinformatics, and artificial intelligence. A basic HMM consists of set of states Θ, a set of initial states $\pi = P[q_1 = \Theta_m]$ such that $1 \leq m \leq N$, a transition probabilities distribution $A = \{a_{i,j}\}$ for transition from states Θ_i to Θ_j, and a set of observations distribution $O = b_j(k)$ where

$$b_n(k) = P[v_k\ at\ t \mid q_t = \Theta_n] \tag{2}$$

for $1 \leq n \leq N$, $1 \leq k \leq M$. The set of states is defined by

$$\Theta = \Theta_1, \Theta_2, \ldots, \Theta_N \tag{3}$$

In the proposed vCell, the HMM states can be characterized as high, medium, and small workloads. These states may vary at various task execution phases such as reading data, performing computation, and writing results.

The next section demonstrates the application of HMM to achieve dynamic resource provisioning and allocation. We define provisioning as the task of creating and configuring the cloud resources for job execution. The task of allocation involves scheduling and submitting jobs to the created resources for execution.

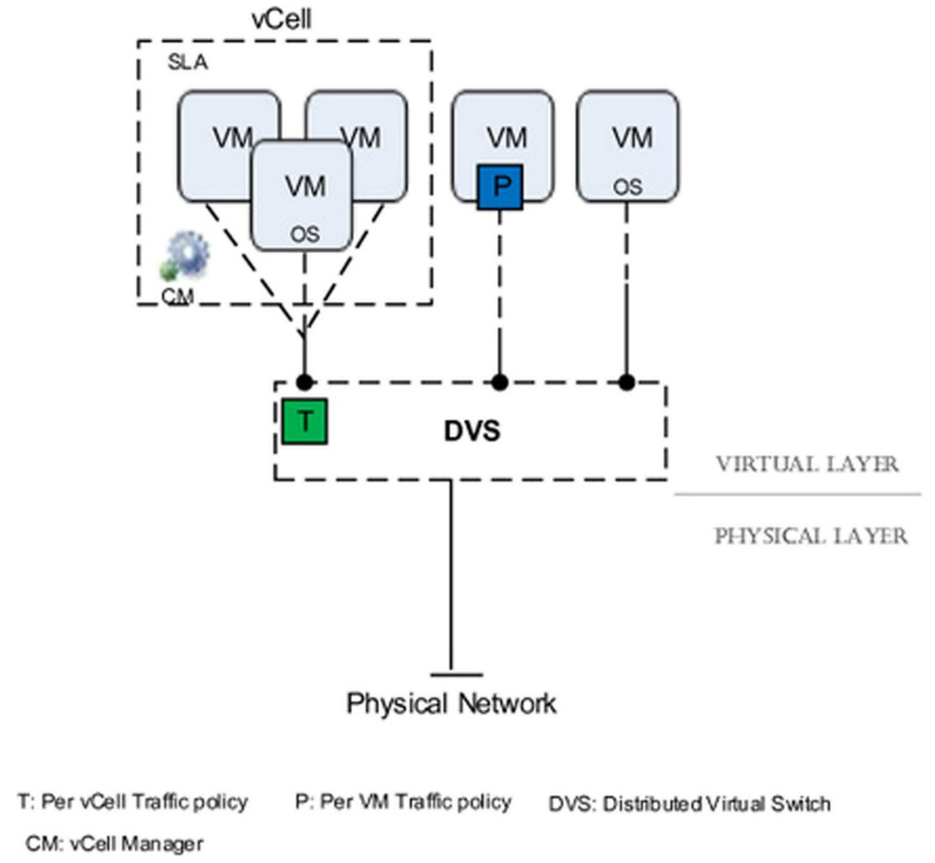

Figure 3 Architecture of the proposed isolation in a virtual infrastructure container using distributed virtual switch (DVS).

Finish time

Our model relies on the finish times (predicted by the vCell manager) of jobs on a virtual machine to optimize the provisioning of virtual resources and allocation of jobs. The finish time F_j^t of job j at any t is computed as:

$$F_j^t = \frac{S_j + \beta}{C_v} + \frac{S_j}{L_c} \tag{4}$$

S_j is the size of job j, C_v is the processing capacity of virtual machine v that the job j is submitted to, L_c is the available link capacity, and β is VM processing policy such that $\beta = 0$ if the policy is single processor allocation policy and β is a value that denotes other workloads at time t and is defined as:

$$\beta = \begin{cases} 0 & \text{if single worker processor} \\ \ell & \text{otherwise} \end{cases} \tag{5}$$

ℓ is the predicted size of jobs concurrent on machine i at time t. For central storage servers (e.g. network file servers or NFS), ℓ is the workload exerted by virtual machines accessing the data stored on the server. Estimating ℓ is particularly challenging due to a number of factors. Firstly, the notion of virtualization means that a large number of VMs sprawl on physical servers. This makes the observable workload values incomplete due to background processes and resource contention. Secondly, task execution exerts a time varying workload on the virtual machine. A task in execution has different workload patterns at various stages of the execution. For instance, the data loading process takes a considerable amount of time in data-intensive allocation after which the execution is a memory intensive process and exerts less demand for IO. Consider a simple model where the task execution overhead is given as:

$$\Psi_i(t) = \Sigma_{j \in J} \left(R_{i,j} + N_{i,j} - E_{i,j} \right) * X_{i,j}(t) \tag{6}$$

where $X_{i,j} = \{0, 1\}$ is a binary variable indicating whether virtual machine i is executing job j at time t, $R_{i,j}$ is the computation overhead of job j on machine i, $N_{i,j}$ is the network overhead incurred during data access, and finally $E_{i,j}$ is the total time spent on the execution of job j at time t.

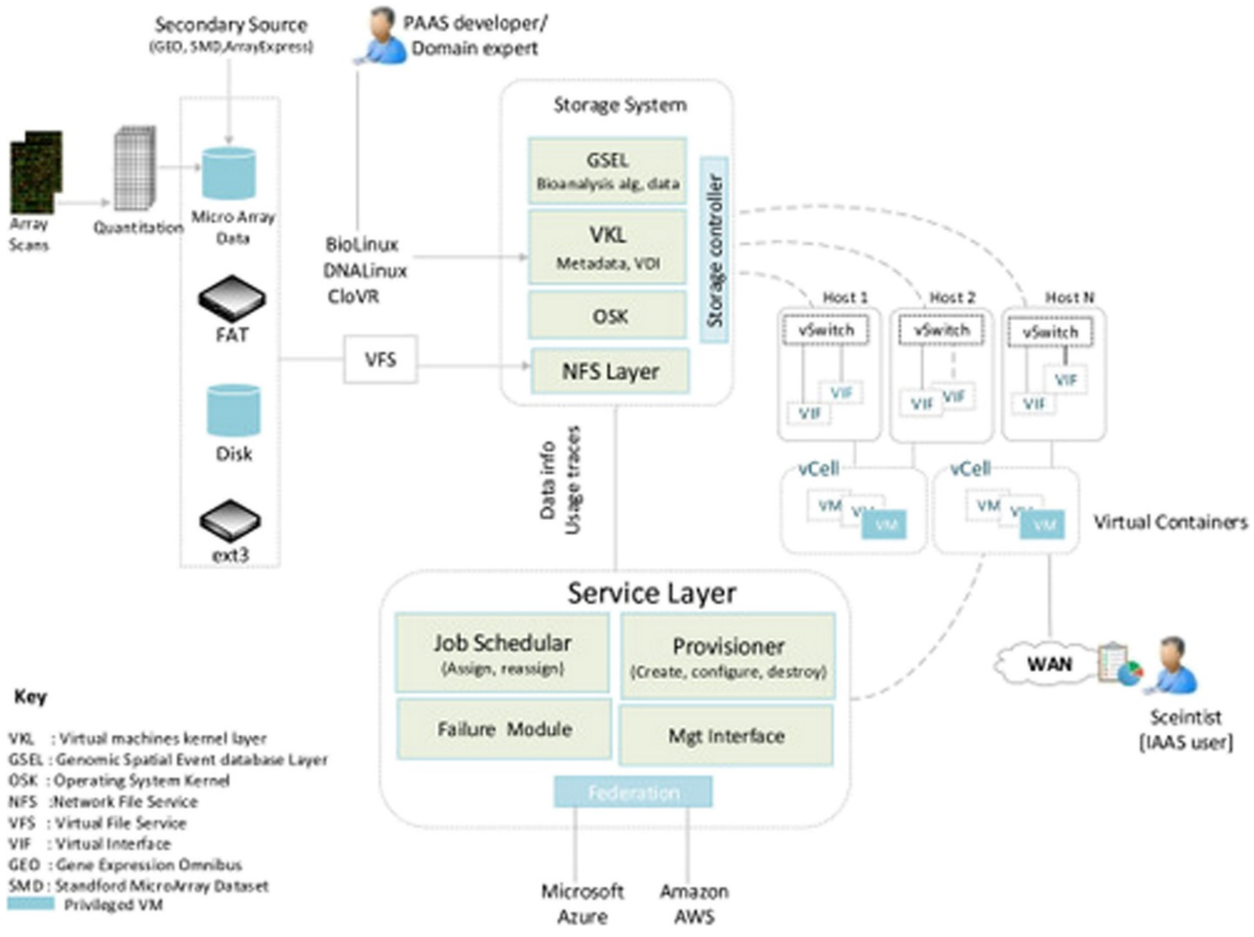

Figure 4 Architecture of resource interactions in Intra Cloud Datacenter (DCN) suitable for large scale data analysis.

To predict ℓ we adopt an inference mechanism based on the posterior distribution of a parameter for any given observation traces, X. Specifically, we adopt the Naive Bayes Classifier (NBC) and Markov Chain states model to predict the jobs concurrent on i at a given time t. NBC is a supervised learning classifier used in data mining [37]. Using the notation $X_{1:t-1}$ to mean all the observations $X_1, X_2, \ldots, X_{t-1}$, we seek to estimate the state Ψ_t at any time t using all the previous observations $X_{1:t-1}$. From Bayesian rule we have the posterior probability for any A, B, and C as:

$$P(A/BC) = \frac{P(B/C)P(B/A)C}{P(B/C)} \tag{7}$$

Applying bayesian recursive property to equation 6 we obtain:

$$P(\Psi_t \mid X_{1:t-1}) = \Sigma_{\Psi_{t-1}} P(\Psi_t \mid \Psi_{t-1}) P(\Psi_{t-1} \mid X_{1:t-1}) \tag{8}$$

Equation 8 depends only on the previous state and the observations at time t-1. For $i \in V$ machines and given transitional states as the workload Ψ, the probability of a VM i having workload $\ell_i(t) = P(\Psi_t \mid X_{1:t-1})$ is thus given by equation 8 irrespective of the observation at t.

Initial VM

We formulate a technique to determine the capacity of VMs for a job submitted. This work uses joint VM provisioning in a container based model. We propose the cost model $f(M_h, P_t, B)$ to estimate the initial capacity C. Given a budget of B units, the cost model estimates the initial capacity C of VMs using time series [8] of similar tasks. We compute the capacity C for each container as a function of the user defined budget B, standard machine hour M_h cost, and the processing time F_j^t obtained from equation 4. The capacity of VMs required is computed as:

$$C = \frac{B(S, \varphi)}{M_h * \sum_{j \in S} F_t^j} \tag{9}$$

φ is the concession, per unit cost invested, that the cell owner is willing to allow and is inversely related to the budget. S is the job size submitted for analysis.

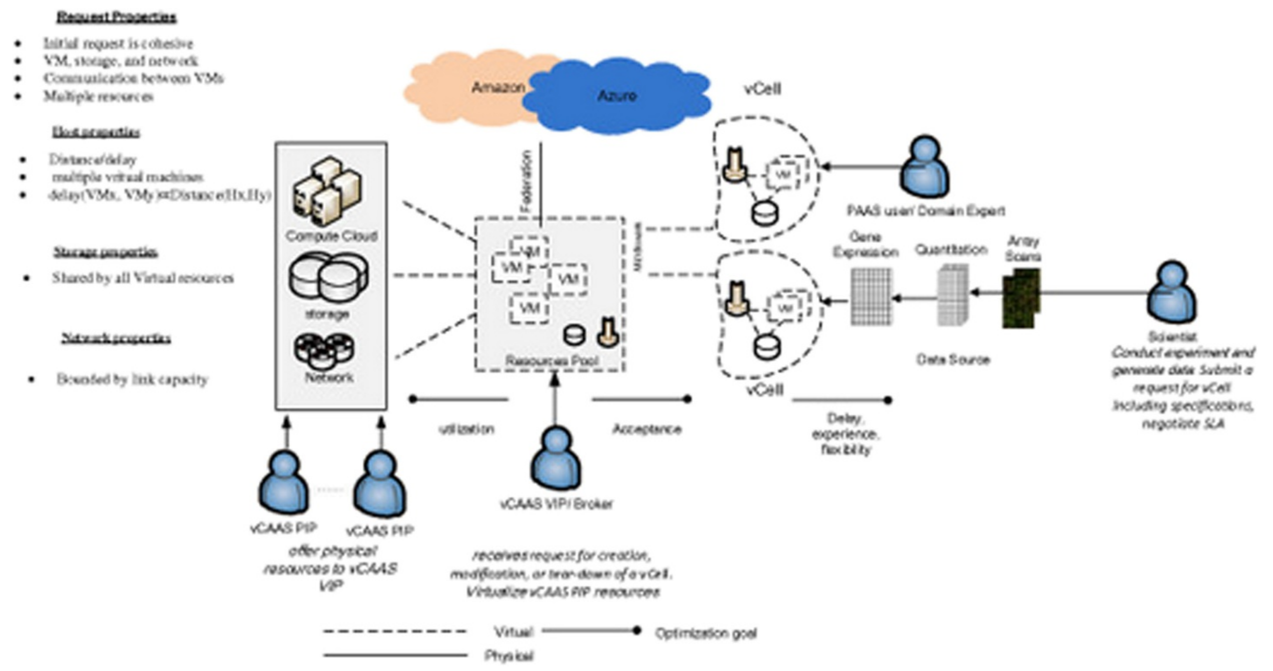

Figure 5 **Interaction between various actors in a data intensive vCAAS.**

Understanding the microarray data

For several years, biologists and life scientists have been using GeneChip technology to analyze gene expression in cells. Affymetrix supplies small glass slide arrays on which many probes of 25 bases of DNA or RNA sequence data are established. The probes are designed so that sets of probes on the GeneChip will test for the expression of particular genes. The data read from one array is stored in one CEL file, which has a standard format but can be either binary or character text which results in a variation in the size of the CEL files. Researchers tend to use several arrays to test different conditions so that one set of experiments which have a connection with each other are stored in related CEL files in a folder called a GSE (GEO Series Experiment).

The microarray data from many experiments are uploaded to public databases such as GEO (Gene Expression Omnibus) [38], for other scientists to use in their research. In this work, some of the experiments which use the Human GeneChip called HG-U133A were downloaded and analyzed. The analysis, carried out using the R statistical language, was to determine whether runs of guanine in the probe sequences (runs of 4 or more 'G's) were producing a significant bias in the gene expression data [14,39].

The microarray data from many experiments are often uploaded to public databases [40] such as GEO and ArrayExpress, for other scientists to use in their research. The microarray data studied in this work has the following characteristics:

- All jobs are submitted at the initial phase of the experiment.
- All CEL files in a GSE folder must be processed to complete a successful analysis.
- Each GSE folder contains between 10 and 700 CEL files each between $4MB$ and $32MB$ in size.
- To process each folder, depending on the size, requires memory capacity in the range 1 GB to 16 GB.
- All data to be processed must be loaded in the main memory.
- Time to process data depends on the number of CEL files and total GSE folder size.
- Data access time varies considerably depending on the size of CEL files in the GSE folder.

Implementing the data analysis container

Our work considers self-service and dynamic algorithms for initial VM size, VM provisioning, and job allocation. The proposed algorithms are implemented in 4 steps. To perform analysis of this type of data, the following steps are required:

Step 1: Read job configuration settings.
Step 2: Estimate the number of VMs for the given data to analyse and provision the required VMs. For the chosen microarray data analysis, this step requires the classification of the data into sub group and the types of virtual machines with

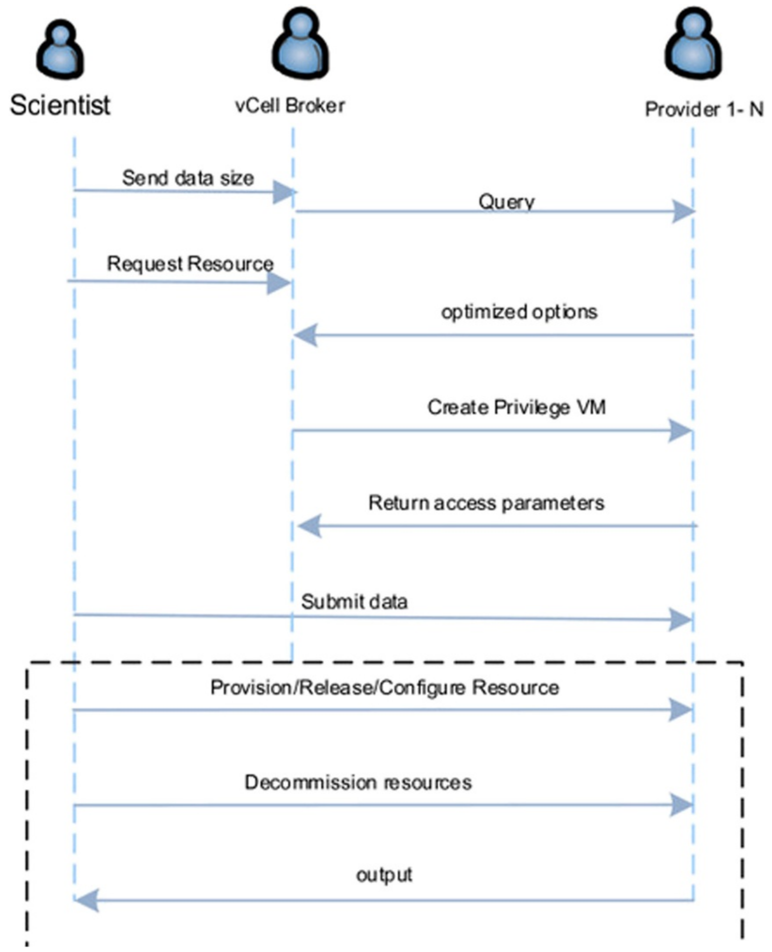

Figure 6 Sequence diagram for the creation and use of a vCell.

required memory for each group created. The dynamic classification provides automation and eliminates the need to manually assign the jobs thereby reducing the delay and, subsequently, the cost associated with manual resource creation and release. This works well for data intensive tasks. However, for other application types with different requirements other than memory, a different scheme is required. In both cases, knowledge of the domain and the historical output trace of previously executed related jobs are valuable inputs to the inference mechanism to determine the relationship between the task requirements and capacity (e.g. bandwidth, memory, CPU) of the VMs.

Step 3: Creation of virtual network (VIF) for the created VM. This stage takes into consideration both the task stage requirement.

Step 4: Configuring the created VMs with required software packages and network address to enable interaction using the VIF.

Step 5: Allocate submitted jobs to the VMs.

Step 6: Release the VMs and scale-down the vCell resources if there are no more jobs to process.

The major challenges for scientists using cloud computing to effectively and efficiently analyse this type of data are steps 2–5. For instance, the time between creating a resource and then submitting the job adds to the total idle time of the VM. This incurs additional cost as a result. Similarly, VMs not released after job execution further incur additional cost as most public clouds charge per hour [41]. The next subsection proposes dynamic mechanisms to achieve these steps without user intervention. Novel provisioning, job allocation, and adaptive clustering algorithms [42] are presented to enable the self-service data analysis.

Virtual machines provisioning

The provisioning problem for any given set of folders J each of size S_i and set of virtual machines M each $VM_j \in M$ of capacity C_j. The problem is to find the 1-to-1 mapping $J_i \longrightarrow M_j$ such that $Si \in Mj$ at any time t.

In a similar way to a biological cell, the strict requirement for all initial functional VMs (control, report, failure, and service console) to be created is enforced; otherwise the vCell request is rejected. All the algorithms considered in this work make use of the holistic view and complement each other to execute the submitted task cohesively.

To estimate the initial VMs' capacity we use equation 9 with $N = 3$ number of states (Θ) to predict the finish time (equation 4). For the data described, we identify the states in terms of workload size as high (Θ_3), medium (Θ_2), and low (Θ_1). Each VM processes a submitted job through all or part of the states. Each state varies depending on the current process. We identify three such distinct observed current processes as read (xR), write (xW), and computation (xC). It is evident as shown in Figure 7, that the average time spent on data access constitutes less than 30% of the total job finish times. This signifies that job processing is read intensive (xR) at the beginning of microarray data analysis and then compute intensive (xP) afterwards. Notice that the data access time depend on the number of VMs concurrently accessing the data stored at the NFS server. The higher the number of VMs accessing the NFS server concurrently, the more workload at the NFS server and hence slower data access time may be experienced by the VMs. In the Figure 7, jobs 9 and 10 seems to be accessing the NFS server at the same time and hence the large data access time. The job allocation strategy proposed in this research is to vary the data access period of the VMs to reduce concurrent access and consequently reduce the data access times.

We choose the initial state in Θ based on the hidden state probabilities distribution $\phi = \phi_i$ for $1 \leq i \leq N$. Tables 1, 2, and 3 shows the initial state probabilities distribution, observation distribution, and transition probabilities respectively for each VM in the vCell. Although the ϕ is chosen based on observable job processing cycle, it has been shown in [36] that the selection and update process in Markov Chain modelling eliminates the initial error over long iteration of equation 8.

LJF-KQ algorithm

To provision the required VMs, we proposed a variation of [30] as Largest Job First on the K Queues (LJF-KQ) strategy. In this scheme (Algorithm 1), we implement a K-queue on-demand provisioning policy that leases VM instances based on K job categories (classified based on size). If we denote $Q_1, Q_2, \ldots, Q_K$ as the queues of large, medium, and small jobs; then the algorithm proceeds as follows. Initially, one VM for each queue type is created on the first fit host starting with Q_1 and then Q_2. Subsequently, one VM per each Q_K job is created until the maximum allowable number of VMs are reached. Idle VMs are assigned new jobs from the job category they are created for. Once all jobs from a particular category Q_n are completed successfully, the resource allocated to the VMs for Q_n is reclaimed and re-allocated to create virtual machine(s) for other job categories in the global queue.

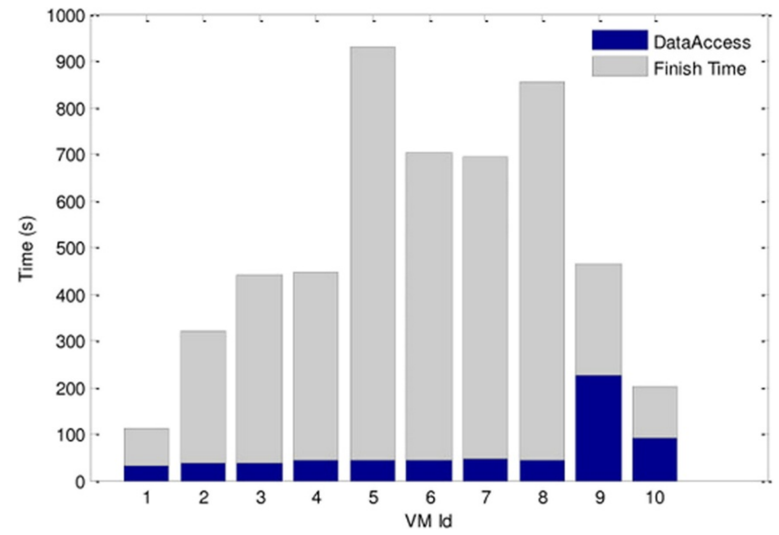

Figure 7 Variation in data access time and total finish time for various jobs running on a virtual machine.

Algorithm 1 LJF-KQ (K Queue Largest Job first)

```
Input: J – job list, trace- previous job completion time, $M_H$ – standard cost of machine hour, C – container capacity requested in terms of bandwidth RAM and MIPS, minC – minimum capacity, $K$ - job queue types, Type – distinct job categories , $C_i$ – capacity of virtual machine i, $B$ - Budget
Output: V – list of VMs created
 1: Set B
 2: T ← trace(Size(J)) {estimate time required to complete job}
 3: $C \leftarrow B/(M_H * T)$
 4: Type = KMeans(J,K)
 5: SORT Type in descending {sort the job categories}
 6: count=0
 7: for t = size(Type) to 1 do
 8:     count=0
 9:     Create virtual machine $V_{count}$ for Type(t) {one VM from each category except last category }
10:     $C \leftarrow C - C_{count}$ {update available capacity}
11:     $V \leftarrow V \bigcup V_{count}$
12:     count= count+1
13: end for
14: while $C > 0$ do
15:     Create virtual machine $V_{count}$ for Type(t) {one VM from each category except last category }
16:     $C \leftarrow C - C_{count}$ {update available capacity}
17:     $V \leftarrow V \bigcup V_{count}$
18:     count= count+1
19: end while
20: Return V
```

Algorithm 2 below shows the pseudo code for the classification using K-Means. The initial cluster K is determined by the available classes of VMs. Amazon AWS, for instance, offers many instance types (small, medium, large) based on capacity (memory, storage, bandwidth) and target optimization.

Algorithm 2 K-Means

```
Input: Training Data - J, job clusters - K,
Output: Classified jobs - Type
 1: n=length(J)
 2: Initialize K clusters centroids $\mu = \mu_1, \mu_2, \ldots, \mu_k \in R^n$
 3: repeat
 4:     for i=1 to n do
 5:         $c^{(i)} = index(1:K) \in \mu$ Closest to $J^{(i)}$
 6:         $Type(J^{(i)}) \leftarrow c^{(i)}$
 7:     end for
 8:     $\Delta = 0$
 9:     for k=1 to K do
10:         $mu_k^{'} = \mu_k$
11:         $\mu_k$ = average mean of points assigned to cluster k
12:         $\Delta = \Delta + \nabla(\mu^{'}, \mu)$
13:     end for
14: until $\Delta = 0$
15: Return Type
```

Table 1 Sample initial state probabilities distribution

Θ_1	Θ_2	Θ_3
0.5	0.3	0.2

LJF-KQ-L algorithm

The Largest Job First on K Queues with Lookup (LJF-KQ-L) algorithm is a variation of LJF-KQ with lookup for finish times. The finish times F is obtained from pilot data. Just like LJF-KQ, a large VM is created for Q_n jobs initially. However, in creating VMs for the next job categories one VM per job is created from Q_{n-1} while the estimated finish time of the jobs is less than the finish time of Q_n. This continues until n = 1. Then the process begins with Q_n again. This continues until the maximum VMs size is reached or vCell capacity is reached. The algorithm can be summarized as follows:

- Classify the submitted jobs based on size and files count using K-Means clustering
- Select a job from the job classes and create the VM for the chosen job size.
- SORT in descending order
- N = 0
- SELECT job category L_N
- Set current large job as L_N
- CREATE one VM for L_N with capacity $C(L_N)$

1. Create VMs from next large job L_{N-1} with capacity $C(L_{N-1})$ until capacity allocated for L_N is reached
2. Set $N = N + 1$
3. Repeat 1–3 until maximum capacity is reached

Algorithm 3 gives the detail steps in the proposed algorithm

Table 2 Sample discrete observations probabilities

State| Operations	xR	xW	xC
Θ_1	0.1	0.6	0.3
Θ_2	0.2	0.4	0.4
Θ_3	0.3	0.3	0.4

Algorithm 3 LJF-3Q-L (K Queue Largest Job first)

```
Input: J – job list, trace- previous job completion time, $M_H$ – standard cost of machine hour, C – container capacity requested in terms of bandwidth RAM and MIPS, minC – minimum capacity, K - job queue types, Type – distinct job categories , $C_i$ – capacity of virtual machine i, B - Budget
Output: V – all VMs created
Ensure: $C > 0$
 1: Set B
 2: T ← trace(Size(J)) {estimate time required to complete job}
 3: $C \leftarrow B/(M_H * T)$
 4: Type = KMeans(J,K)
 5: SORT Type in descending {the job categories}
 6: N=length(Type)
 7: count=0 , n=N
 8: while $n > 0$ do
 9:     count=0
10:     Create virtual machine $V_{count}$ for Type(n) {one VM from each category except last category }
11:     $C \leftarrow C - C_{count}$ {update available capacity}
12:     $V \leftarrow V \bigcup V_{count}$
13:     $cummB \leftarrow V_{count}$
14:     count= count+1, sum=0, $\eta$=n-1
15:     while $\eta > 0$ do
16:         while $C > 0$ AND $sum < C_{cumB}$ do
17:             Create virtual machine $V_{count}$ for Type($\eta$) {one VM from this category }
18:             $sum = sum + C_{count}$
19:             $C \leftarrow C - C_{count}$ {update available capacity}
20:             $V \leftarrow V \bigcup V_{count}$
21:             count= count+1
22:         end while
23:         $\eta$=$\eta$-1
24:     end while
25: end while
26: Return V
```

Job allocation algorithm

Next we present the algorithms for job allocation considered in this work. The use of container based resource provisioning enables the sharing of statistics between the vCell manager and other resources in the vCell. This interaction is computationally and network transfer wise expensive if implemented for the whole datacentre. At the boot, all VMs register with the resource database in the cell manager module. The virtual machines are allocated bandwidth to satisfy the current job requirement (computed from time series). At the end of the data access time, the bandwidth allocated is reconfigured to allow other VMs to utilize the container's overall capacity.

Table 3 Sample state transition probabilities

States	Θ_1	Θ_2	Θ_3
Θ_1	0.1	0.4	0.5
Θ_2	0.4	0.4	0.2
Θ_3	0.2	0.4	0.4

SJF-KQ

The shortest job first algorithm on K queues (SJF-KQ) is a variation of Shortest Job First (SJF). In this approach, jobs are ordered in decreasing order of size and submitted to the suitable category, K, of VMs created during the provisioning stage. This algorithm executes the submitted jobs based on the resource queues implemented in the provisioning phase.

SJF-KQ-L

The shortest job first algorithm on K queues with Lookup (SJF-KQ-L) is a variation of SJF-KQ. However, the expected finish time of each job is utilized to vary the data access period of the jobs. If the data access time of the job i is T_i, then the input/output overhead at storage server contributed by i at each point in F_i- T_i is zero. This variation on data access is exploited to reduce the overheads associated with data access by the virtual machines during jobs execution. The algorithm carefully schedules the job execution of VMs to ensure that the number of concurrent VMs accessing the NFS server is reduced. This way, the makespan is consequently reduced.

FCFS-KQ-L

The First Come First Serve algorithm on K queues with Lookup(FCFS-KQ-L) is a variation of SJF-KQ-L that is based on the widely used First Come First Serve (FCFS) discipline instead of the SJF.

Implementation and results

This section demonstrates the implementation stages of adopting a holistic view to resource management and job allocation in performing large data analysis. The approach taken is to implement well-known ordering disciplines - FCFC and SJF - in provisioning of cloud resources and then apply the container-based model to measure the benefits of our approach. Java code and R script are implemented to provide the features of the perceived vCAAS for the analysis of a large data intensive application.

Infrastructure setup

A room is dedicated to the cloud facility that served as a testbed cloud within the School of Computer Science and Electronic Engineering at the University of Essex. The following are the hardware infrastructure (Figure 8) deployed for the experiment consisted of:

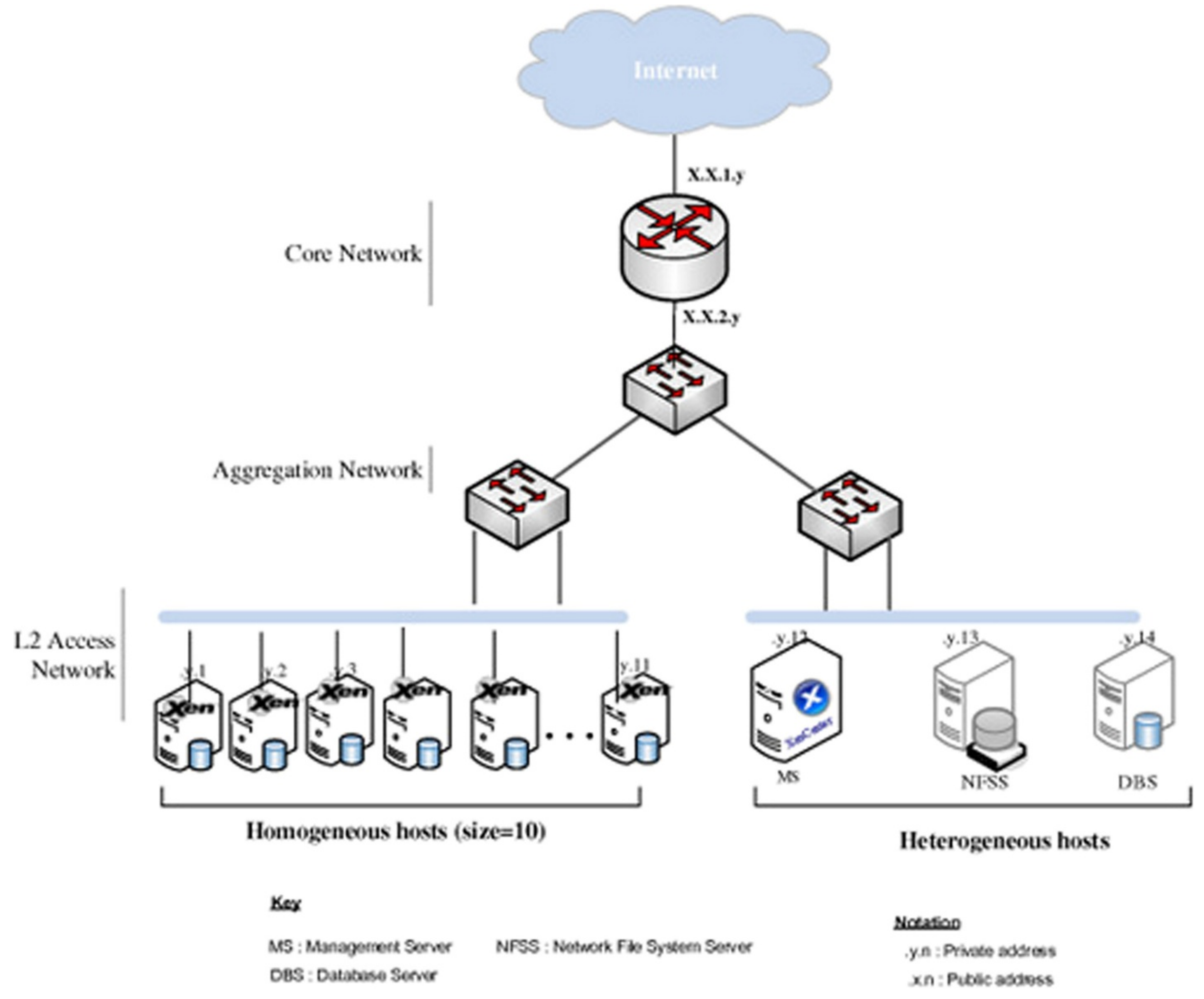

Figure 8 Architecture of the testbed deployed for data analysis experiment.

1. Ten homogeneous 64 bit Dell OptiPlex systems with 4 *GB* RAM, core 2 Duo processor running at 3.33*GHz*, and 300 *GB* hard disk storage capacity. Each computer has 4 virtual cores per processor. The internal prototype experiment cloud is therefore created on the 40 virtual cores of total processing capacity and 80 *GB* of total RAM. We deploy XEN Cloud Platform (XCP) [43] on all the 10 servers for hosting virtual machines. Deploying XCP enable us to combine all the capacities into one large pool of memory and processing capacity.
2. One 64 bit Dell OptiPlex system with 8 *GB* Ram and core 2 Duo processor running at 3.33*GHz* is set aside for the NFS server (NFSS). The computer has a total usable storage capacity of 20*TB* comprising the internal hard disk and the external storage array directly attached to the server.
3. One 64 bit Dell OptiPlex system with 8 *GB* RAM, core 2 Duo processor running at 3.33*GHz* and 4 virtual cores, and a hard disk storage capacity of 300 *GB* is set aside for the management server (MS) which also serves as the Middleware server.
4. One 64 bit Dell OptiPlex system with 8 *GB* Ram, core 2 Duo processor running at 3.33*GHz* is set aside for storing results of gene expression in a MySQL database server (*DBS*). The *DBS* has a capacity of 300 *GB* hard disk and 4 virtual cores per processor.

The experiment set-up to demonstrate the data analysis is depicted in Figure 8. In the experiment set-up the following features are employed:

- Cheap commodity network and IT hardware are deployed for the experiment.

- Physical hosts are connected in 3-layered hierarchical network topology (Figure 8) comprising aggregation, edge, and access layers.
- Basic modules presented in Section 'Container-based cloud framework' are created and packaged in a privileged virtual machine.
- The privileged VM is created and configured with functionalities for resource provisioning and dynamic job allocation. This packaged virtual machine is created in the requested container.
- Each virtual service cell is equipped with basic service functionalities suitable for the microarray data analysis.
- Experiment data stored in various storage media are made available via a standard network file system.
- The functionalities available in the vCell estimate finish time based on previous time series statistics of finished job and dynamically adjust the operations of various components.
- Provisioning and job allocation algorithms are tested using the value $K = 3$ for the job queues.

The classes to enable the main datacentre functionalities (Figure 9) are written in java and installed in the MS. The classes to implement the storage service are also written in Java and installed in the NFSS as set of Java objects. And finally a stand-alone MySQL Server 5.1. is installed at the DBS to save the results of the microarray data analysis.

vCell implementation

The code to realize the proposed virtual infrastructure container is divided into five sets of modules. Each module is created to perform interdependent task. The modules are for vCell creation, vCell request creation, storage access, gene expression identification, and the result writer. vCell creation is performed by the datacentre main classes. The datacentre main module provides all the functionalities for parsing submitted vCell request in VXDL format, create and initializes the privilege VM (pVM) to enable the function of vCell Manager. The functions of provisioning, allocation, and report writing are projected into the pVM. The provisioning is invoked by the pVM which starts the operation of the provisioner.

The vCell resources (VM, VIF) creation modules reside in pVM enabled with a subset of the service layer. Resource provisioning, task classification and allocation, resource release and update, and inference learning are all performed by this module. The class diagram for this module is shown in Figure 9. The provisioning process starts with an estimation of the required initial VMs. Initial VMs capacity for the vCell is computed from the implementation of equation 9. The capacity is then used as input to XEN API libraries to create the required VMs. The Provisioner classifies the submitted GSE data into groups using an implementation of dynamic K-Means strategy shown in algorithm 2. After the classification, the function *getSuitableHost*() is called to invoke the host service wrapper class (HostServiceWrapper). The HostServiceWrapper replies with the set of available physical hosts and their available capacities. pVM then chooses the most suitable as guided by the SLA parameters in vCell's submitted request.

After the selection of suitable host, the required Xen Api (XEN) VM class is invoked to create all the required initial VMs. The VIF class instance is then invoked to create a network of links and virtual interfaces. One virtual interface is created for each network definition in the VXDL request file. Using the initial task inference from the dynamic Markov chain model, an initial virtual interface is selected as the default for communication. The allocated bandwidth to a VM is constantly updated by the pVM to reflect the various phases of job execution at the VM. This process ensures that available bandwidth in the vCell are properly budgeted.

Each created VM is started via the implementation of a call to the VM.start method. The VMs are configured with a standard socket to listen, on a specific port, for any incoming job submitted by pVM. The class RScript starts the R script, if it is not running, and set the current working directory of the R workspace to the directory where the data to be analysed is saved. At the end of computation the results are written back to the VM class which subsequently write the final result to the MySQL database server.

We implement virtual file service (VFS) functionalities in combination with allocation and control to provide an effective and flexible storage server. Figures 8 and 9 show the realization of the storage access module. Our approach attempts to minimize the IO overhead overtime caused by VM sprawl. Each VM is enabled with a microarray data analysis algorithm and accesses the required files in a "just in time" policy. This approach allows ease in relocation of jobs since only the required data is copied and the relocation to any idle resource is easily achieved.

The pVM periodically checks all running instances and decides, based on the status information (failed, running, idle, halt) obtained, whether to reassign the job. The functionalities provided include: initiate termination of a VM, report reclaim resources to resident vCell manager, and initiate creation of new virtual resource. This way, the job status can be determined and, where necessary, relocation of the job to a new instance initiated. The set-up determines job status by computing completion time as a function of GSE folder size, available memory in allocated VM, and data transfer delay. After the expected finished time, VM is marked inaccessible and the job running in the VM marked as failed. If status is failed, the module notifies the provisioning module which then destroys the

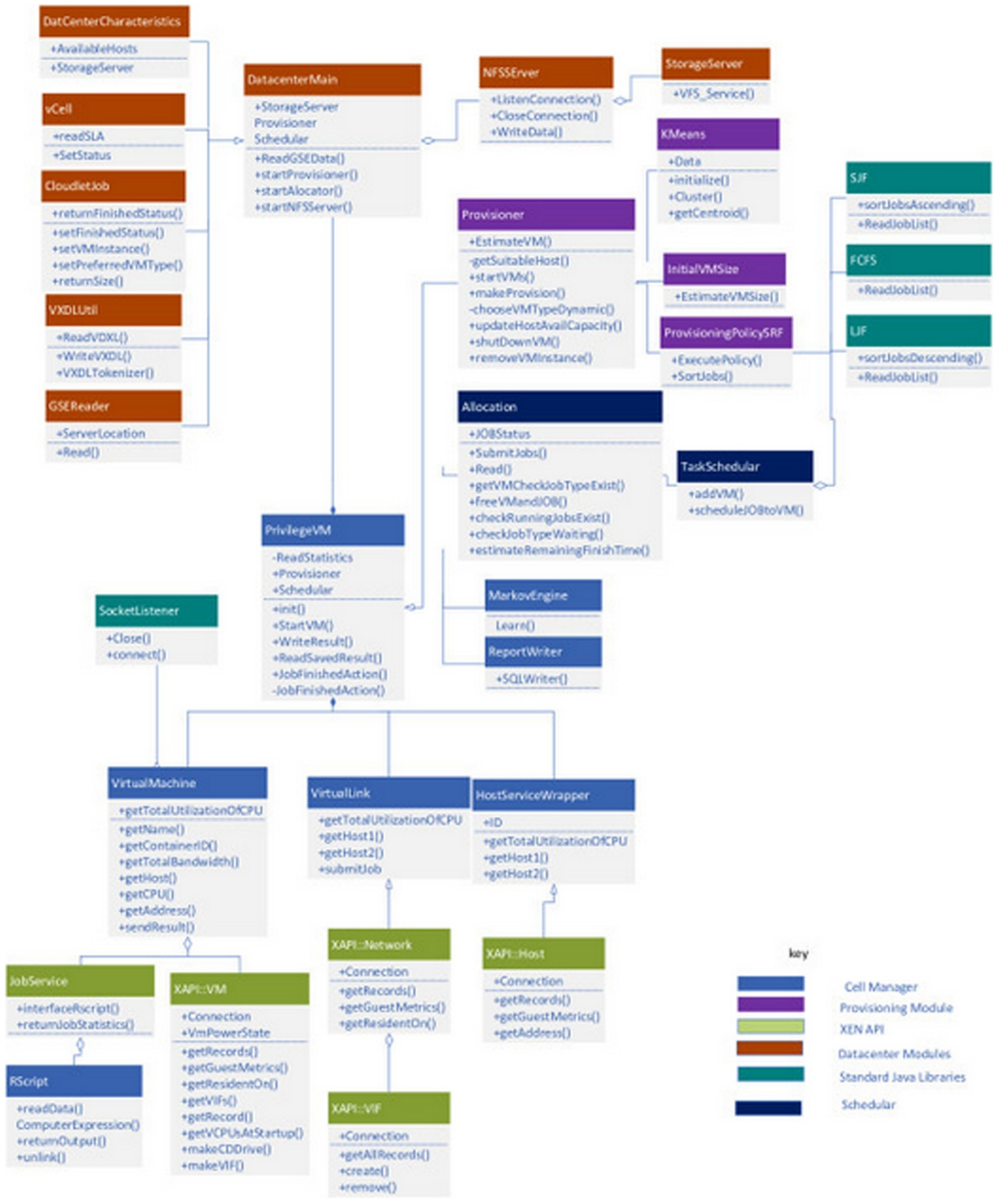

Figure 9 vCAAS classes and functions for data analysis.

inaccessible VM, assigns the destroyed VM's resources to create new VM, and finally reassigns the failed job to the newly created VM.

Experiment results

In this section we present the results of our data intensive experiment on a private cloud. Our work investigates the impact of adopting cohesive operational behaviours among the VMs to exploit the differences in data access times and implement effective resource provisioning and job scheduling. Each virtual machine requests bandwidth of a certain size to satisfy the job submitted. After the duration of data access, the virtual machines bandwidth is reduced to the basic bandwidth. This is achieved by inserting new action table entry in the software-defined enabled virtual switch introduced in Section 'Container-based cloud framework'. The residual bandwidth from the reconfiguration is made available for other VMs.

Our first experiment investigates the performance of our proposed provisioning algorithms. The algorithm that utilizes the predicted finished time lookup during provisioning is compared using three well-known [30] on-demand algorithms for virtual machines provisioning. Figure 10 show that our proposed algorithms reduced the thrashing rate - frequency of creation and destruction of virtual machines. High thrashing rate increases the workload on a provisioner and the instability of the data analysis process as resources are created and released. Since cloud services are normally charged per hour, the creation and release of virtual machines incur additional overhead cost. Hence, a small value for the thrashing rate is required to maintain a stable and, consequently, cost effective job execution. We use First Come First Serve (FCFS), Largest Job First (LJF), and Shortest Job First (SJF) to demonstrate (Figure 10) that predicting the finish time and taking the prediction into consideration during the classification of jobs into groups during resource provisioning phase reduces the thrashing rate.

In all the experiment, we set the value of K = 3 for the algorithms presented in Section 'Container-based cloud model for bioinformatics'. For example example, LJF-KQL becomes LJF-3QL. The result in Figure 10 shows that the algorithms ('Thrash overhead 3QL') implemented with a lookup outperformed those without lookup ('Thrash overhead 3Q'). In the experiment, a maximum of ten virtual machines and one privilege VM instances are instantiated per vCell. The budget size for the chosen experiment is set to $100. The experiment involved the analysis of 30 *GB* of microarray data. The values on the y-axis in Figure 10 shows the number of times a VM is released and new one created to accommodate new analysis job.

In the Figure 10, all our three algorithms that enhanced common provisioning disciplines (FCFS, SJF, and LJF) with a group classification and finished time lookup outperformed those without such enhancement. This is possible as each VM in the vCell operate alongside members of the vCell as a complementary component. Note that achieving such enhanced provisioning is made possible due to the small size of vCell. Implementing the same

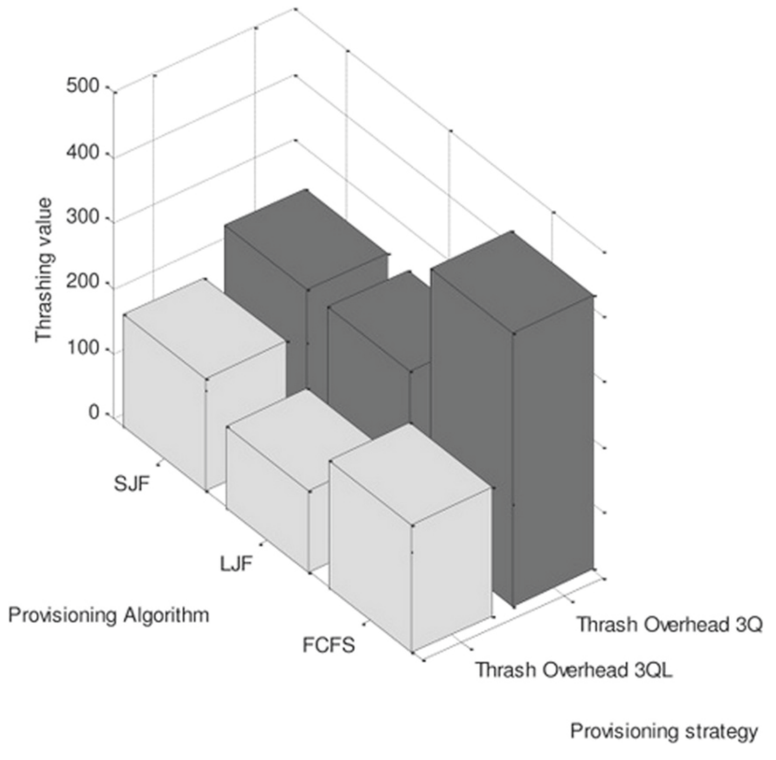

Figure 10 Result comparing thrashing rate between our proposed algorithms using common provisioning algorithms.

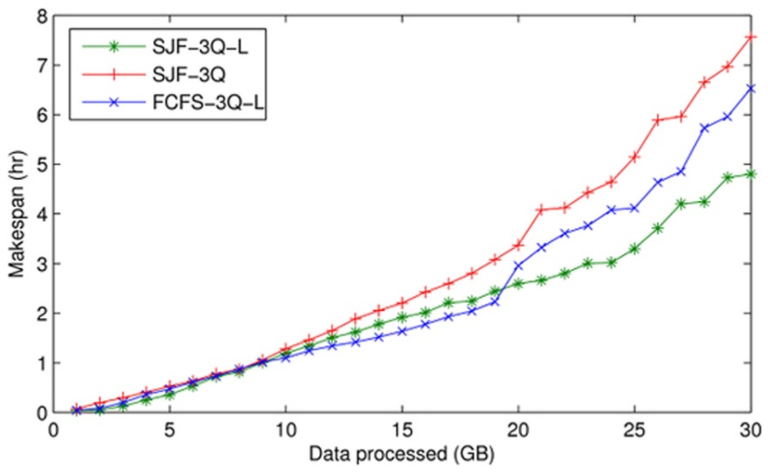

Figure 11 Result comparing the job makespan between our proposed job allocation algorithm and other algorithms.

finished time lookup and grouping of jobs to complement each other at the whole datacentre level will amount to large computation in the domain of NP-hard problem.

We continue with a comparison between our modified versions of SJF task allocation algorithms as SJF-3Q and SJF-3Q-L respectively. Figure 11 shows the result obtained from analyzing up to 30 *GB* of microarray data. We first defined the makespan as the time difference between the start and finish of the data analysis and include time to read the required data, performs compuation to identify gene expression, and write the output result to the DBS. In the figure, SJF-3Q-L outperformed the other two algorithms with shorter makespan at various job sizes. We attribute this higher performance to the ability of our vCell manager to assign jobs based on expected finished times.

We then investigate the impact of using our proposed holistic view of cloud resources on the cost of data analysis. As demonstrated in Figure 12, the cost of analysing the data is lower for algorithm utilizing the finish times of jobs on virtual machines. We attribute the performance strength of SJF-3Q-L on two features:

- the small size of VM thrashing reduces the cost of resource usage since VMs are only created gradually as resources are released by the large number of small VMs.
- by carefully utilizing the jobs statistics from the report interface module, we can allocate jobs in a way that reduces concurrent data access and improves the performance.

In summary, the combined results in Figures 10, 11, and 12 highlights that holistic view of resources improved the performance of well-known algorithms for resource provisioning and job allocation. Quantitatively, the cost of performing analysis is reduced by 50% at 15 GB of data

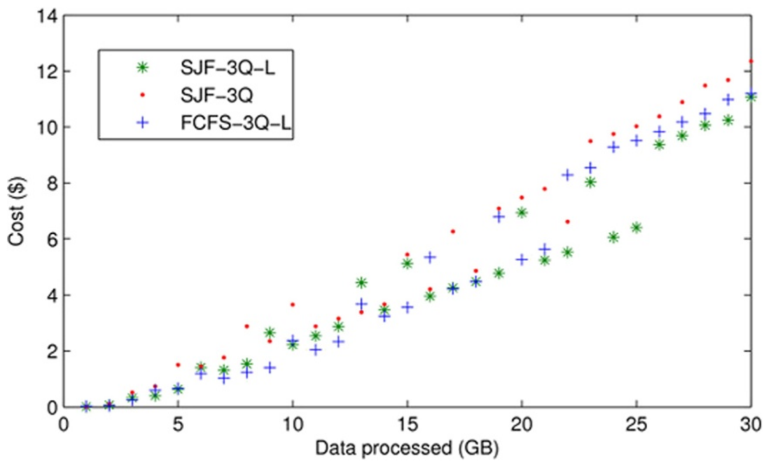

Figure 12 Cost comparison among various job allocation algorithms.

analysis. The makespan also reduces by more than 1 hr at 15 GB of data analysis.

Conclusion

There is a general perception [44,45] that the next wave of cloud services to be dominated by PAAS value added services. The virtual container described in this work is a step toward this advancement. In our case, the value added service is created by the bioinformatician and packaged as a virtual machine. The scientist performing the experiment request this virtual machine and other required resources as a self-service and dynamic container. Our work demonstrates the strength of this next generation cloud framework in performing cost effective analysis of microarray data on commodity hardware. Flexibility, dynamic configuration, and elasticity were enabled by creating a self-service infrastructure container which allows the scientist performing the experiment to submit an abstract description of requirements. Furthermore, the cloud framework proposed in this work allows VMs to operate in concert with each other and with the enabling logic. The dynamic feature of the model reduces the need to understand technical cloud computing concepts.

The virtual container presented in this work is enabled with Markov Chain learning and prediction that allows the container to manage itself using previous observations from job execution traces. We use the automation capability to estimate initial VMs' capacity without the intervention of a user.

This article demonstrates the concept in a prototype experiment cloud built on commodity hardware. The cloud environment is created using XEN Cloud Platform (XCP). The proposed privileged virtual machine is equipped with necessary XAPI compliant java modules.

A significant difference between the strategy described in this work and existing clouds is the holistic view of the resource. Also in the proposed framework, use of an observed pattern of data analysis is applied to automate the whole data analysis process. Using the variation in instant virtual machine bandwidth requirements, our proposed algorithms improved the performance and led to considerable reduction in cost at a performance that guarantees the same experience as commercial cloud services. Although this work focused on a data intensive cloud application, the same logic can easily be extended to other cloud applications. In the future, our work aims to implement the same container model for parallel and distributed cloud applications.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

IKM developed the algorithms to dynamically provision virtual resources and allocate submitted jobs in vCAAS. IKM built the cloud environment to test the proposed cloud model. AMO and APH provided the domain specific expertise of the chosen biological data and helped develop the algorithm for the data analysis. SW verifies the applicability of the model in the chosen application area and supervised the development and testing of the models proposed. All authors read and approved the final manuscript.

Acknowledgement

IKM wish to thank Abubakar Tafawa Balewa University Bauchi Nigeria for awarding him the scholarship to study in the UK. The authors would like to express their thanks to Hugh P. Shanahan and Farhat N. Memon for the original data analysis algorithm [14], and to Graham J.G. Upton who contributed to it.

Author details

[1]School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, Essex, UK. [2]Department of Mathematical Sciences and Biological Sciences, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, Essex, UK.

Received: 5 November 2013 Accepted: 14 April 2014
Published: 22 May 2014

References

1. Michael A, Armando F, Rean G, Joseph AD, Katz RH, Andrew K, Gunho L, David AP, Ariel R, Matei Z (2009) A view of cloud computing. Commun ACM 53(4):50–58
2. Srirama S, Batrashev O, Vainikko E (2010) Scicloud: scientific computing on the cloud. In: Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing. IEEE Computer Society, pp 579–580
3. Hines MR, Deshpande U, Gopalan K (2009) Post-copy live migration of virtual machines. ACM SIGOPS Oper Syst Rev 43(3):14–26
4. Al-fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A (2010) Hedera: dynamic flow scheduling for data center networks. In: NSDI, pp 19–19
5. Benson T, Akella A, Maltz DA (2010) Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th Annual Conference on Internet Measurement. IMC '10. ACM, New York, pp 267–280
6. Wang G, Andersen DG, Kaminsky M, Kozuch M, Ng TSE, Papagiannaki K, Glick M, Mummert L (2009) Your data center is a router: the case for reconfigurable optical circuit switched paths. Comput Sci Dep:62
7. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 1(1):7–18
8. Meng X, Isci C, Kephart J, Zhang L, Bouillet E, Pendarakis D (2010) Efficient resource provisioning in compute clouds via vm multiplexing. In: Proceedings of the 7th international conference on autonomic computing. ACM, pp 11–20
9. Vinothina V, Shridaran DR, Ganpathi DP (2012) A survey on resource allocation strategies in cloud computing. Int J Adv Comput Sci Appl 3(6):97–104
10. Ferrer AJ, Hernández F, Tordsson J, Elmroth E, Ali-Eldin A, Zsigri C, Sirvent R, Guitart J, Badia RM, Djemame K (2012) Optimis: a holistic approach to cloud service provisioning. Future Generat Comput Syst 28(1):66–77
11. Banerjee P, Friedrich R, Bash C, Goldsack P, Huberman BA, Manley J, Patel C, Ranganathan P, Veitch A (2011) Everything as a service: Powering the new information economy. Computer 44(3):36–43
12. Musa IK, Stuart W (2014) Multi objective optimization strategy suitable for virtual cells as a service. In: Innovations in bio-inspired computing and applications. Springer, pp 49–59
13. Zorov DB, Kobrinsky E, Juhaszova M, Sollott SJ (2004) Examining intracellular organelle function using fluorescent probes from animalcules to quantum dots. Circ Res 95(3):239–252
14. Shanahan HP, Memon FN, Upton GJG, Harrison AP (2012) Normalized affymetrix expression data are biased by G-quadruplex formation. Nucleic Acids Res 40(8):3307–3315
15. Schatz MC, Langmead B, Salzberg SL (2010) Cloud computing and the dna data race. Nat Biotechnol 28(7):691
16. Field D, Tiwari B, Booth T, Houten S, Swan D, Bertrand N, Thurston M (2006) Open software for biologists: from famine to feast. Nat Biotechnol 24(7):801–804
17. Krampis K, Booth T, Chapman B, Tiwari B, Bicak M, Field D, Nelson KE (2012) Cloud biolinux: pre-configured and on-demand bioinformatics computing for the genomics community. BMC Bioinformatics 13(1):42

18. Dudley JT, Butte AJ (2010) In silico research in the era of cloud computing. Nat Biotechnol 28(11):1181–1185
19. Bajo J, Zato C, de la Prieta F, de Luis A, Tapia D (2010) Cloud computing in bioinformatics. In: Distributed Computing and Artificial Intelligence. Springer, pp 147–155
20. Autosomes Chromosome X (2012) An integrated map of genetic variation from 1,092 human genomes. Nature 491:1
21. Kienzler R, Bruggmann R, Ranganathan A, Tatbul N (2012) Incremental dna sequence analysis in the cloud. In: Scientific and statistical database management. Springer, pp 640–645
22. Stein LD (2010) The case for cloud computing in genome informatics. Genome Biol 11(5):207
23. Schatz MC (2009) Cloudburst: highly sensitive read mapping with mapreduce. Bioinformatics 25(11):1363–1369
24. Matsunaga A, Tsugawa M, Fortes J (2008) Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In: eScience 2008, eScience'08. IEEE fourth international conference on. IEEE, pp 222–229
25. Goecks J, Nekrutenko A, Taylor J (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol 11(8):R86
26. Langmead B, Hansen KD, Leek JT (2010) Cloud-scale rna-sequencing differential expression analysis with myrna. Genome Biol 11(8):R83
27. Banerjee P, Friedrich R, Bash C, Goldsack P, Huberman BA, Manley J, Patel C, Ranganathan P, Veitch A (2011) Everything as a service: powering the new information economy. Computer 44(3):36–43
28. Jarzab M, Kosiński J, Zieliński K, Zieliński S (2012) User-oriented provisioning of secure virtualized infrastructure. In: Building a national distributed e-infrastructure–PL-Grid. Springer, pp 73–88
29. Díaz FEO, Gómez SG (2012) 4caast technical value proposition. 4CaaSt consortium. http://4caast.morfeo-project.org/wp-content/uploads/2011/02/ValueProposition_Whitepaper.pdf, Accessed: 2013-09-30
30. Genaud S, Gossa J (2011) Cost-wait trade-offs in client-side resource provisioning with elastic clouds. In: Cloud computing (CLOUD), 2011 IEEE international conference on. IEEE, pp 1–8
31. Adam C, Stadler R (2005) Adaptable server clusters with qos objectives. In: Integrated network management, 2005. IM 2005. 2005 9th IFIP/IEEE international symposium on. IEEE, pp 149–162
32. Buyya R (1999) High performance cluster computing. Prentice Hall PTR, Upper Saddle River. 99017906 edited by Rajkumar Buyya. ill. ; 25 cm. Includes bibliographical references and indexes. v. l. Architectures and systems – v. 2. Programming and applications.
33. Wood T, Gerber A, Ramakrishnan KK, Shenoy P, Van der Merwe J (2009) The case for enterprise-ready virtual private clouds. Usenix HotCloud https://www.usenix.org/legacy/events/hotcloud09/tech/full_papers/wood.pdf, Accessed: 2011-01-3
34. Koslovski GP, Primet PV-B, Charão AS (2009) VXDL: virtual resources and interconnection networks description language. In: Networks for grid applications. Springer, pp 138–154
35. Sherwood R, Gibb G, Yap K-K, Appenzeller G, Casado M, McKeown N, Parulkar G (2009) Flowvisor: a network virtualization layer. OpenFlow Switch Consortium, Tech. Rep
36. Rabiner LR (1989) A tutorial on hidden markov models and selected applications in speech recognition. Proc. IEEE 77(2):257–286
37. Webb GI, Boughton JR, Wang Z (2005) Not so naive bayes: aggregating one-dependence estimators. Mach Learn 58(1):5–24
38. Barrett T, Troup DB, Wilhite SE, Ledoux P, Rudnev D, Evangelista C, Kim IF, Soboleva A, Tomashevsky M, Edgar R (2006) NCBI GEO: mining tens of millions of expression profiles–database and tools update. Nucleic Acids Res 35(Database issue):760–765
39. Memon FN, Owen AM, Sanchez-Graillet O, Upton GJG, Harrison AP (2010) Identifying the impact of G-quadruplexes on Affymetrix 3' arrays using cloud computing. J Integr Bioinform 7(2):111
40. Ball CA, Brazma A, Causton H, Chervitz S, Edgar R, Hingamp P, Matese JC, Parkinson H, Quackenbush J, Ringwald M (2004) Submission of microarray data to public repositories. PLoS Biol 2(9):317
41. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M (2009) Above the clouds: a berkeley view of cloud computing. Technical Report No. UCB EECS-2009-28, 2009–200928
42. Chinrungrueng C, Sequin CH (1995) Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. IEEE Trans Neural Network 6(1):157–169
43. Williams DE (2007) Virtualization with Xen (tm): including XenEnterprise, XenServer, and XenExpress. Syngress
44. Garcia-Gomez S, Jimenez-Ganan M, Taher Y, Momm C, Junker F, Biro J, Menychtas A, Andrikopoulos V, Strauch S (2012) Challenges for the comprehensive management of cloud services in a paas framework. Scalable Comput: Pract Exp 13(3)
45. Natis YV, Lheureux BJ, Pezzini M, Cearly DW, Knipp E, Plummer DC (2011) Paas road map: a continent emerging. Gartner Res. Gartner (Inc)

doi:10.1186/2192-113X-3-5
Cite this article as: Musa *et al.*: Self-service infrastructure container for data intensive application. *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:5.