# CLOUD RESOURCE MANAGEMENT USING A HIERARCHICAL DECENTRALIZED FRAMEWORK

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2022

Abdul Rahman Hummaida

Department of Computer Science

# Contents

**Word Count: 76182**

# List of Tables

# List of Figures

# Abstract

CLOUD RESOURCE MANAGEMENT USING A
HIERARCHICAL DECENTRALIZED FRAMEWORK
Abdul Rahman Hummaida
A thesis submitted to The University of Manchester
for the degree of Doctor of Philosophy, 2022

Cloud providers (CPs) build and operate large scale data centres that contain numerous computing resources that are typically virtualized and require a level of orchestration of the shared resources. With an increased demand for cloud computing resources at a lower cost by end-users, CPs need to increase the efficiency of their infrastructure usage. To achieve this, CPs aim to increase resource utilisation and lower operational costs, typically the energy to administer, run and cool computing resources. A promising approach to increase the efficiency of infrastructure usage is to adapt the assignment of resources to workloads. This can be used, for example, to apply a policy that conserves energy by combining workloads and enabling CPs customers to meet their performance objectives. The mapping of workloads to data centre resources can be viewed as being carried out by two abstract components, Management Algorithm (MA) and Management Framework (MF). The MA is responsible for deciding how workloads are assigned to infrastructure resources. At the same time, the MF enables the MA to execute by providing standard functionality, such as the scope of the infrastructure being managed and aggregation of metrics that will allow the MA to make decisions. Several architectural solutions have been presented for MFs. However, these tend to be centralized and may suffer in their ability to run the MA at scale and support data centres with thousands of physical nodes. Decentralized approaches solve the scalability problem but have a limited view of resources across the data centre,

which reduces the opportunity to remap resources across a larger scope of the infrastructure. Several techniques are used for MAs, with heuristics being a common choice. However, heuristics' performance depends on multiple factors, including the statistical patterns of workload demands, and if the underlying scenario changes, heuristics may start to perform poorly.

This thesis is grounded on the hypothesis that solving the scalability challenge in mapping workloads to resources starts by addressing scalability in the MF. We propose a novel scalable hybrid MF and demonstrate this to improve the ability to meet performance objectives and provide a global view of the infrastructure through empirical evaluation. To address the challenge with heuristic MAs, we propose a reinforcement learning-based MA that can learn a policy to dynamically balance achieving Service Level Agreements, achieve high CPU utilization, and remove the need to use defined CPU utilization thresholds. We combine the proposed MF with the proposed MA and demonstrate this outperforms heuristic approaches in reducing service level agreement violations and provides high CPU utilisation through empirical evaluation.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

# Acknowledgements

I am fortunate and grateful to have people that helped and supported me in a multitude of ways, and I have found them invaluable during this PhD. I would like to express my gratitude to them.

My supervisors, Professor Norman Paton and Professor Rizos Sakellariou, believed in me and supported me in pursuing this on a part-time basis. They gave ample guidance, continuous support, empathy, and insightful advice on my research.

My parents encouraged me to pursue this before I knew I wanted to. My family, partner and friends, gave me the push to continue when times got hard.

Thank you for the love, care, and endless support.

# Chapter 1

# Introduction

Cloud computing is an established paradigm for providing on-demand services to various end-users, including computing and storage infrastructure. Compared to traditional customer-managed infrastructures, customers access infrastructure resources and control software from a Cloud provider (CP). The research firm International Data Corporation (IDC) [69] forecasts cloud spending to grow at a five-year compound annual growth rate of 9.6%, with public clouds accounting for the majority of the growth, shown in Figure 1.1.

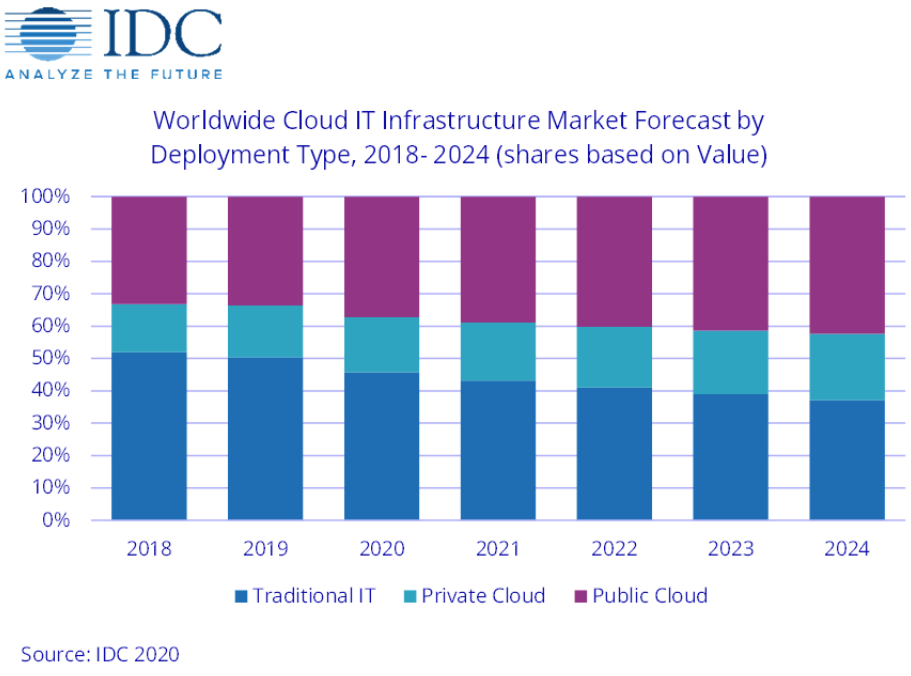



Figure 1.1: Global Cloud usage forecast [69]

Cloud Providers (CPs) provide access to resources on-demand and enable provisioning through APIs or web portals. Figure 1.2 shows a representation of cloud resources and customer access. Resources are typically pooled and shared between customers, with a layer of orchestration that separates individual customer usage. Physical resources are abstracted through virtualization technology into computing, memory, storage and networking with a logical separation of these resources and typically presented as a Virtual Machine (VM). CPs provide managed services such as orchestration mechanisms and logging of customer workloads to simplify usage of the infrastructure. Access control mechanisms provide the ability to determine how users and web traffic access the infrastructure. By pooling access to resources, CPs can consolidate multiple underutilised resources into fewer physical resources and save on energy consumption and operational costs. Customer workloads are stochastic, and CPs need to re-optimize the infrastructure regularly to provide high levels of availability and reliability.

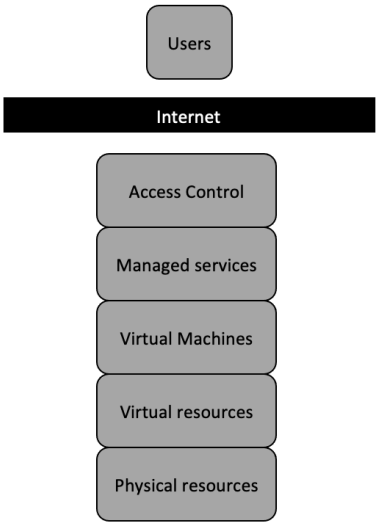



Figure 1.2: Cloud resources

Standard cloud deployment models are: private, public and hybrid. A *private cloud* is typically built for and used by a single organisation, usually due to security constraints. *Public clouds* offer infrastructure resources and managed services on varying payment models over the internet to the general public. Current examples of public cloud providers include Amazon AWS, Google, Microsoft, IBM, and Rackspace. Customers benefit from public clouds by utilising increasing resources through a subscription or usage model without capital or maintenance expenditure. Public clouds

have multi-tenancy, where multiple customers coexist, and services owned by numerous providers are co-located in a single data centre [160]. Physical resources are shared and dynamically assigned to multiple customers. A *hybrid* cloud is a mixed usage of private and public clouds and enables a customer to expand out to a public cloud to run workloads that are deemed less sensitive to the organisation.

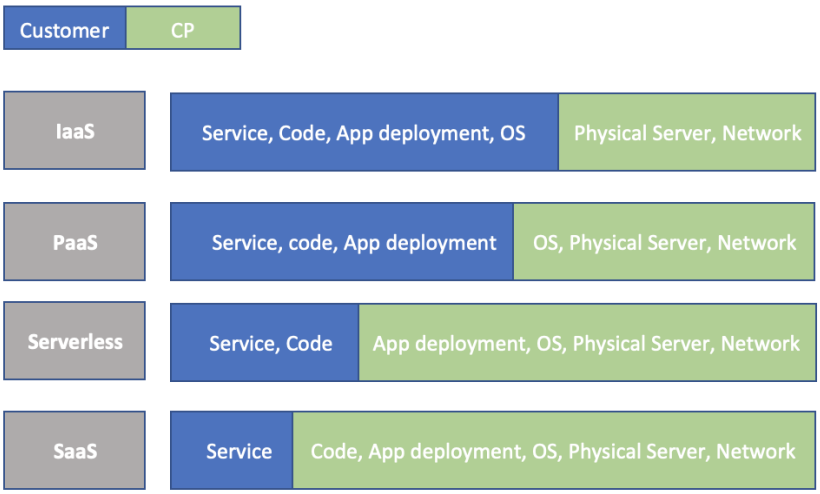



Figure 1.3: Public Cloud service models and Customer/Provider responsibilities

Standard service models in public clouds are: Infrastructure as a Service (*IaaS*), Platform as a Service (*PaaS*), *serverless* and Software as a Service (*SaaS*) and are shown in Figure 1.3. This shows decreasing levels of customer responsibility from IaaS towards SaaS. During the emergence of cloud computing, IaaS was the initial service model, with compute, networking and storage exposed as a capability. Customers combine these with their defined Operating System (OS), code and service to form their infrastructure. In this model, customers rent the underlying physical infrastructure, which the CPs provide. End users can log onto a web portal on the IaaS to install OSs on VMs, deploy software components such as libraries and databases for a given application. While IaaS provides customers with flexibility, it requires a high level of technical knowledge to optimise and manage the virtual infrastructure presented by the CP [96]. Compared to traditional on-premise infrastructures, IaaS removes all of the management and maintenance of physical infrastructure and provides the ability to add additional resources in minutes compared to days or weeks. Examples of CPs providing IaaS include AWS EC2, Google Compute Engine and Microsoft Azure.

Platform as a service (PaaS) is a service model where CPs provide hardware, software stacks and runtime environments for application development. Customers have

control over the development environment, including configuration. The CP hosts the hardware and software on its infrastructure and gives customers an extra level of abstraction compared to IaaS, removing maintenance of application stack, runtime environments, operating systems and databases. Example PaaS providers include AWS Elastic Beanstalk, Force.com and Google App Engine.

In the *serverless* service model, customers create specific parts of their intended application through designated code functions, and the CP handles scaling and exchanging of data between different functions. Therefore serverless computing is a service model that allows customers to run event-driven and granularly billed applications without addressing the operational logic of constructing applications [143]. There is an overlap between serverless computing and PaaS, and both aim to create abstractions to build software applications. Serverless computing provides a higher level of abstraction as software developers focus on the business logic of applications with no visibility on the orchestration and deployment mechanism onto cloud resources or the management of the physical servers and network.

In *SaaS*, the customer uses general applications that are typically accessible through multiple devices that are connected to an internet connection. SaaS is the highest level of abstraction for customers. They focus on configurations specific to their usage, such as end-user privileges, branding, available features and customisation of business logic workflows, with all infrastructure management being handled by the CP and or a software vendor.

CPs manage the cloud infrastructure, including physical nodes, storage and network connectivity, and typically present these as virtualized resources, with a VM as a common resource on a pay-per-use basis. CPs manage the VM creation process and assign VMs to physical nodes, and a single node may host multiple end-user VMs. The method of configuring VMs and mapping them to nodes has been widely researched [50, 100, 83, 116, 131, 130, 133, 27, 32], and can be viewed as initial VM placement, state detection and migration [24]. VMs contain CPU, memory, network, storage and workloads with variable demands that are created through an *initial VM placement* process. CPs regularly re-evaluate existing mappings due to VMs not meeting their SLAs, or to remap resource assignments to improve resource utilisation and reduce usage costs by performing *VM migration*. Another objective for CPs is to satisfy their customers by achieving service level agreements (SLAs), which specify the quality and scope of the service provided to customers. With the increased popularity of cloud

systems, the energy consumption and environmental impact have given rise to more research into efficient resource management that can provide workloads with the required computational capacity and reduce energy waste [26]. However, the management of infrastructure resources and applying CPs objectives is a complex process. We classify this management process into two dimensions, *Management Algorithm (MA)* and *Management Framework (MF)*, shown in Figure 1.4. The MA is responsible for deciding how incoming workloads are assigned to infrastructure resources by regularly assessing the satisfaction of such assignments in achieving a given SLA. The time complexity of the MA influences the frequency of this assessment; the lower the complexity, the more frequently the algorithm can be executed. The MF enables the MA to execute by providing standard functionality, such as hierarchy level management, the scope of the infrastructure under control or aggregation of utilisation metrics. The combined functionality of the MA and MF results in workloads executing on infrastructure nodes and dynamic reassignment of workloads to resources.

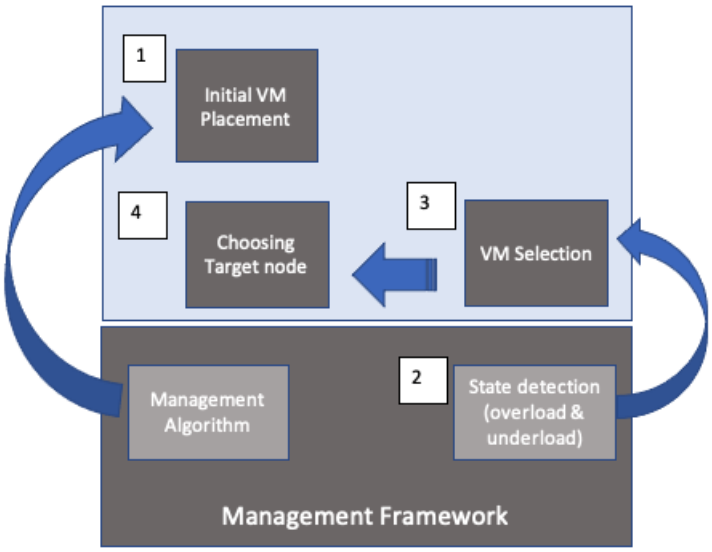



Figure 1.4: Cloud Resource Management process

## 1.1 Scalability challenges in resource management

Cloud resource configuration can be applied dynamically to remap resource assignments. Examining management frameworks (MFs) used in resource management shows these can be centralized, hierarchical or decentralized [90], and have different scalability properties. To assess MFs, we use the definition of scalability as the ability

of the system to sustain increasing workloads by making use of additional resources [91]. Centralized MFs use an engine with a global view of the entire managed infrastructure and map resources across the whole infrastructure. Hierarchical MFs typically divide the infrastructure into multiple sections, with a decision engine in each section, which creates a centralized MF for each section. Decentralized MFs distribute the management of the infrastructure without a centralized controller. Both hierarchical and decentralized architectures are a form of distributed management architectures. Centralized, hierarchical and decentralized MFs have their merits. Centralized has a global view, hierarchical has increased scalability compared to centralized, and decentralized has no central controller and has been shown to scale well and manage a large number of nodes. The resource mapping process is expressed in the Management Algorithm (MA) in the context of the MF, and as the size of the managed resource increases, the mapping process becomes more complex. Centralized MFs are common in the literature [163, 159, 24, 34, 85]. These rely on a single source of truth that performs all the phases required for resource mapping, including monitoring, calculating the schedule, and executing the resource mapping. During the decision making and application of resource mapping, centralized MFs may not be able to maintain QoS properties and react quickly to violations [117]. A centralized MF may compromise the mapping quality to lower the execution time for decision making to support scalability. Additionally, the continuous resource utilization collection can be expensive to process and further impacts scalability [126]. Hierarchical architectures are centralized MFs, that divide the infrastructure into multiple sections, with a decision engine in each section. While this improves scalability compared to centralized MFs, as the size of the managed infrastructure increases, the approach suffers similar challenges. Decentralized MFs enable resource management decisions to be taken through the collaboration of multiple decision making entities, and can scale to manage a large number of nodes [90, 151, 126]. VM consolidation is the process of placing as many VMs in fewer nodes using VM migration. While decentralized approaches have been able to reach VM consolidation performance of 75% CPU utilisation [126, 90], they can be limited by the smaller view of the infrastructure, which reduces the number of nodes that can be taken into consideration during VM consolidation.

Key to this thesis is a proposal for a hybrid MF, which is a scalable hierarchical decentralized MF that overcomes the weaknesses of centralized, hierarchical and decentralized approaches, and achieves improved QoS metrics. Through empirical evaluation, we show the merits of this hybrid approach, which has the scalability of

decentralized approaches combined with the benefits of a larger infrastructure view in hierarchical approaches.

## 1.2 Motivation & Research Aims

Analysis of existing cloud resource management systems shows that while extensive research on MAs has been conducted, these are primarily centralized MFs, thus having the expected scalability limitations. This thesis focuses on issues related to the provisioning and mapping of cloud resources to workloads, from the CP perspective, to improve the scalability of the mapping process in the MF. Success can then be measured through improved QoS metrics and reduced SLA violations. The thesis is grounded on the hypothesis that *cloud management systems can be designed to apply global control and achieve high scalability through a hybrid management architecture combined with a self-adapting MA*. Based on the research hypothesis, the broad objectives of the research are as follows:

(**Obj1**): Characterise and compare the composition and attributes of cloud management systems through a literature review.

(**Obj2**): Design a new hybrid MF that retains the advantages of decentralized hierarchal MFs and reduce their combined disadvantages. In particular, retain the high scalability of decentralized MFs and the global view of hierarchal MFs.

(**Obj3**): Design a framework to evaluate the capabilities of the new MF by integrating multiple heuristic MAs into the hybrid MF.

(**Obj4**): Design a self-adapting MA that utilises the hybrid MF escalation features and achieves low SLA violations.

(**Obj5**): Design the self-adapting MA to remove the need for operators to set CPU threshold and achieve improved CPU utilization, compared to existing heuristics.

This thesis focuses on CPs objectives in a PaaS context, and presents the results of work undertaken towards providing scalable and efficient approaches for the MF and MA. We focus on large scale cloud environments, with thousands of VMs that run CPU intensive applications such as web servers. The resources assigned to the VM and the number of incoming requests determine the response time experienced by end users. The aim is to enable VMs to meet their response time and achieve a cost

efficient utilization of the cloud infrastructure. We model applications as a single VM, however, the proposed approaches can be extended to manage applications made up of multiple VMs and other application types, such as a batch process.

## 1.3 Methodology

This section describes the research methodology used in this project and consists of a literature review, model development and validation and evaluation. This is shown in Figure 1.5.

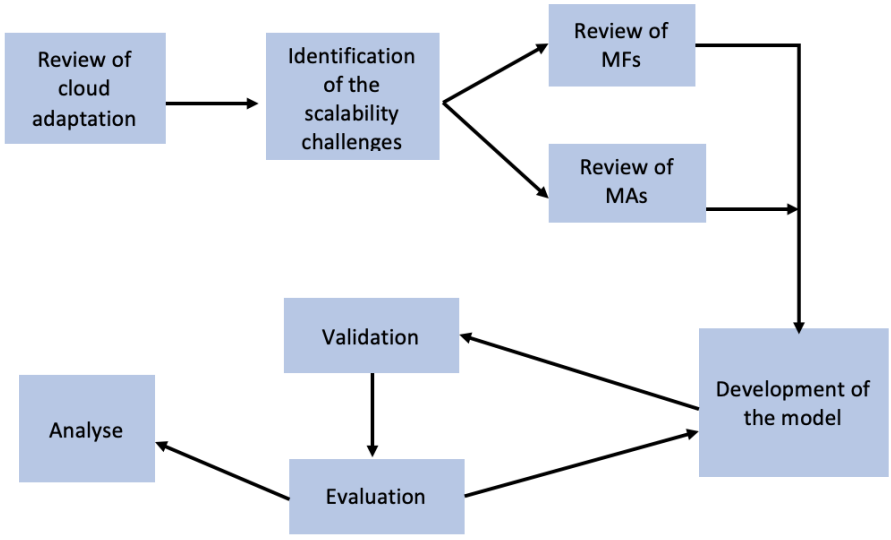



Figure 1.5: Research methodology

### 1.3.1 Literature Review

The first objective (Obj1) of this research project is to review and analyse the related work to identify the properties of cloud resource management systems. This enabled the identification of further research opportunities, with many proposals in the literature focusing on MAs. This highlighted a gap in how scalability is achieved in these systems, with decentralized MFs being able to achieve scale but without the larger scope view in centralized and hierarchical MFs.

The literature review method was carried out throughout the PhD project, and each of the published papers presented in this thesis has a literature review focusing on the specific research objective.

### 1.3.2 Development of the model

This stage starts with objective two (Obj2) and investigates the development of a novel hybrid MF that retains the advantages of decentralized and hierarchical MFs. Subsequently, Obj3, Obj4, Obj5 are addressed in the following sequence:

- Investigate the properties of hierarchical and decentralized MFs.
- Investigate a decentralized state dissemination approach that includes decentralized and hierarchical nodes.
- Investigate a decentralized cooperation protocol that drives communication between the decentralized and hierarchical nodes.
- Investigate failure recovery and leader election for the hybrid MF.
- Investigate a suitable simulation environment to validate and evaluate research objectives.
- Investigate the possibility of separating MF and MA and integrating existing MAs from the literature.
- Investigate the possibility of a decentralized Reinforcement Learning MA and incorporating it in the Hybrid MF.
- Investigate the possibility of the MA learning a policy to invoke VM migration, instead of a threshold-based approach.

### 1.3.3 Validation and Evaluation

The Validation stage takes Obj2, Obj3, Obj4, Obj5, and implements them in DCSim [138], which is an extensible simulation framework for cloud data centres. We carry an empirical evaluation with controlled experiments for each validated objective and compare the results with existing MFs and MAs on the dimensions of SLA achievement, energy consumption, and ability to scale with increasing nodes in the infrastructure.

## 1.4 Contributions

The primary contributions of this thesis are illustrated in Figure 1.6 and show the relationship between the research objectives, included papers and contributions. In summary, the contributions are:

1. **Adaptation in Cloud Resource Configuration: A Survey**. A survey of resource reconfiguration in a cloud context including a definition for cloud adaptation and classification that we use to survey the literature. The survey highlights approaches and techniques used to adapt cloud resource configuration, and properties of MAs and MFs. The survey identifies three open research challenges: characterising the workload type, accurate online profiling of workloads, and building highly scalable adaptation mechanisms. The survey fulfils Obj1.

2. **SHDF - A Scalable Hierarchical Distributed Framework for Data Centre Management**. A scalable architecture for data centre infrastructure management, which can manage a large cloud data centre spanning thousands of nodes. This architecture retains the benefits of both hierarchical and decentralized MFs. To the best of our knowledge, this MF is the first hybrid hierarchical decentralized framework that enables nodes to operate in a decentralized manner, combining this with a hierarchical escalation of migration and consolidation of VMs across large sections of the infrastructure. This enables MAs to achieve higher scalability through decentralization and the opportunity to improve consolidation performance through a broad infrastructure view of hierarchical systems. Additionally, the decentralized approach in the hybrid MF enables MAs to have a lower time complexity in mapping VMs to nodes and exploring more optimal options. The design and empirical evaluation of the hybrid MF fulfil Obj2. The proposed MF has low management complexity. Each node is autonomous and performs its own decision making, and uses an efficient gossip protocol to exchange states with other nearby nodes. The proposed hybrid architecture includes a virtual layout of nodes (overlays), that does not affect the physical layout of the infrastructure. The construction of overlays is envisaged as a simple process through an administrator portal that would accompany the proposed hybrid MF, where the size of overlays would be specified. This can be used in the initial instantiation of the infrastructure or as an adjustment of an existing setup.

3. **A Hierarchical Decentralized Architecture to enable Adaptive Scalable Virtual Machine Migration**. To evaluate the proposed MF, we build on the work in [99] and implement similar MAs within our simulation environment. We additionally compare the performance of the MAs in hybrid, hierarchical, decentralized and centralized MFs. All MAs retain their SLA performance properties when running in the hybrid MF, and some exhibit higher SLA performance due to a reduced search space and autonomous properties in the hybrid MF. This demonstrates the feasibility of separating the MF and MA, the flexibility of the hybrid MF, and the ability to integrate it with other MAs to investigate cloud resource management. This evaluation fulfils Obj3.

4. **Scalable Virtual Machine Migration using Reinforcement Learning**. To address the issue with heuristic-based MAs, which cannot adapt the policy used for resource mapping, we propose a reinforcement learning MA, which can integrate well into our hybrid MF and achieve fast convergence and lower SLA violations. A reward function helps the MA learn a VM to node CPU utilisation mapping, which reduces SLA violations. This adapting MA fulfils Obj4.

5. **Dynamic Threshold Setting for VM Migration**. Heuristic MAs typically use threshold-based overload detection to decide when a node is stressed and when to migrate a VM. The challenge with this approach is it requires domain expertise to set the threshold. We propose an MA capable of learning when to migrate a VM without an operator set CPU threshold. Additionally, a reward function penalises migrations that overachieve SLA targets, resulting in the MA achieving SLA targets with reduced energy consumption and high CPU utilization. This MA fulfils Obj5.

## 1.5 Thesis Structure

This thesis is presented according to the guiding principles of the University of Manchester journal format [4]. It contains papers published or submitted for publishing. Their content does not appear in the table of contents, list of tables, or list of figures. The core contribution is a collection of five published and in submission papers, which have been produced during this PhD project. The remainder of this thesis is organised as follows.

**Chapter 1** introduces the problem domain, the motivating research problem in the scalability of MF and MA performance, the research hypothesis, the objectives behind the research and contributions in this thesis.

**Chapter 2** provides a background on the technologies used for resource management systems. It highlights the complexities of managing cloud systems and the competing objectives for infrastructure providers.

**Chapter 3** provides related work in cloud resource management and the dimensions in MFs and MAs. For MFs, it explores the three common architectures: centralized, hierarchical and decentralized. For MAs, it explores allocation techniques, objectives and resources considered during cloud adaptation. The chapter addresses Obj1 leading to contribution 1.

**Chapter 4** presents a collection of published papers and some that are still under review. These are summarised as follows:

**Paper 1:** Addresses Obj1 leading to contribution 1, [65] Hummaida, A. R, Paton, N.W, Sakellariou, R. Adaptation in cloud resource configuration: a survey. Journal of Cloud Computing 5, 1 (2016), 1–16.

**Paper 2:** Addresses Obj2 leading to contribution 2, [66] Hummaida, A. R, Paton, N.W, Sakellariou, R. SHDF - a scalable hierarchical distributed framework for data centre management. In 2017 16th International Symposium on Parallel and Distributed Computing (ISPDC) (July 2017), pp. 102–111.

**Paper 3:** Addresses Obj3 leading to contribution 3, [64] Hummaida, A. R, Paton, N.W, Sakellariou, R. A Hierarchical Decentralized Architecture to enable Adaptive Scalable Virtual Machine Migration. In submission to Concurrency and Computation: Practice and Experience (CCPE).

**Paper 4:** Addresses Obj4 leading to contribution 4, [67] Hummaida, A. R, Paton, N.W, Sakellariou, R. Scalable Virtual Machine Migration using Reinforcement Learning, in the Journal of Grid Computing.

**Paper 5:** Addresses Obj5 leading to contribution 5, [7] Hummaida, A. R, Paton, N.W, Sakellariou, R. Dynamic Threshold Setting for VM Migration, in ESOCC 2022.

In all papers, Abdul R Hummaida contributed the proposal of the main idea, research development, research planning, literature review, writing, evaluation and analysis of the results. Norman W Paton and Rizos Sakellariou supervised the work, contributed to the ideas and proofread all papers. In accordance with the journal format, published papers are presented as they appear in print.

**Chapter 5** provides a critical analysis of related work to the papers included in Chapter 4. Each of the papers has a related work section, and this chapter extends this and provides a detailed comparison. The chapter concludes the thesis and suggests directions for future work.

Figure 1.6 illustrates the relationship between the included papers, research objectives and contributions.

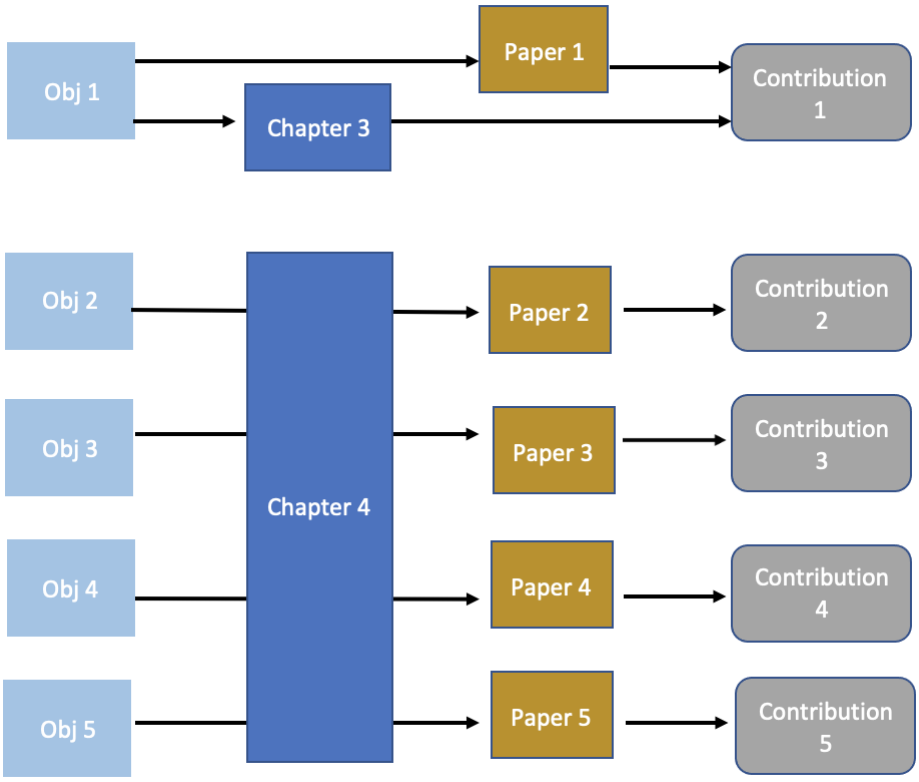



Figure 1.6: Relationship between the research objectives, included papers and contributions.

# Chapter 2

# Problem Background

## 2.1 Introduction

This chapter introduces concepts that are building blocks for this research, including the key technologies that enable the operation of cloud environments. We focus on the areas related to our research topics, particularly virtualization technologies and how Cloud Providers (CPs) adapt resource configuration to achieve their commercial objectives.

## 2.2 Virtualization

Virtualization is an abstraction of physical resources into virtual or logical resources, enabling multiple pieces of software to share a single physical machine. It creates a virtual, rather than an actual, version of a resource. A key example of virtualization is hardware virtualization, where the physical capabilities are presented as multiple logical resources such as a virtual CPU, virtual RAM, virtual network. This enables the creation of a Virtual Machine (VM) where these virtual resources appear as physical resources to all the software running inside the VM. Virtualization allows a single physical node to emulate the behaviour of multiple nodes, with the possibility to host different operating systems on the same node. Thus, a VM is a software implementation of a physical node that can run the software the same way as the physical node would. VMs running on top of a host operating system (OS) execute in an isolated virtual address space and run at a lower privilege than the hosting OS. The hosting OS also exposes virtual disks, virtual CPUs, and virtual network cards.

Virtualization is a fundamental building block that enables cloud computing by facilitating sharing of physical resources. Virtualization enables consolidation, where multiple OSs that may have been running on underutilized physical nodes can be virtualized and moved to fewer physical nodes, reducing CP's capital expenditure and operating costs.

To customers, virtualization offers multiple benefits, including functional isolation, where users can have privileged access within their VM without access to other VMs or the physical node. Virtualization also enables customers to provide environments to deploy and test software faster than physical nodes that need to be acquired, provisioned and connected. While virtualization allows workload consolidation and increases resource utilisation, each VM still runs a full copy of an OS inside the VM. Containerization, on the other hand, addresses this challenge through OS-level virtualization [97]. A container holds the resources needed to run a particular application. It provides a layer of abstraction, with each container behaving like an independent operating system. Each container has its own space of process IDs, its memory, virtual CPUs and a private file system [52]. There is little difference between running in a container or a VM to a customer application. Examples of container services in the public cloud include Amazon Fargate and Google Kubernetes Engine.

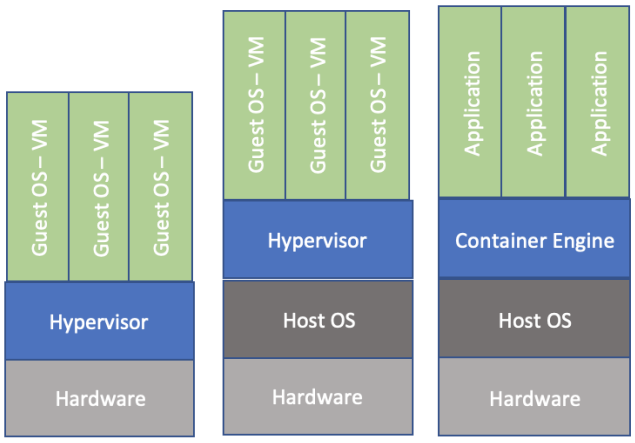



Figure 2.1: Hardware and OS Virtualization

To enable virtualization and control it, the hypervisor [21] is the software that facilitates the creation of abstraction and separation between VMs. Hypervisors act as the equivalent OS and are responsible for scheduling resources and hosting guest OSs [77]. A key component in hypervisors is the Virtual Machine Monitor (VMM), which

captures privileged instructions from guest OSs running on the hypervisor, e.g., network access, and emulate their behaviour by using the physical hardware [72]. The VMM handles exceptions generated by the physical hardware and passes these back to the requesting VM. The VMM also manages CPU virtualization by running most instructions natively and controls specific privileged instructions. This means the performance is almost as good as native code running directly on the hardware [68]. Bare metal, or type 1 hypervisors, run on the physical node and create an abstraction that virtualizes the node devices, as shown in Figure 2.1a. Type 2 hypervisors run on top of an OS, with the OS acting as a host for the hypervisor, as shown in Figure 2.1b. The type 2 hypervisor layer runs as a privileged layer within the hosting OS, and itself can then host other OSs as individual VMs. VMs can run in one of two modes: fully virtualized or paravirtualization. In full virtualization, the VM is provided with a fully compatible interface onto the hardware by the VMM, and the VM requires no modifications or particular setup. Paravirtualization is a technique where the guest OS is explicitly modified to run on a hypervisor. A set of drivers on the OS call an interface on the hypervisor to optimise performance compared to complete virtualization [72].

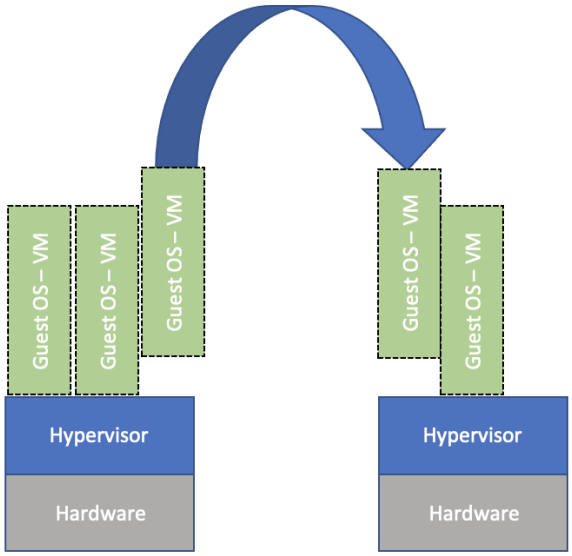



Figure 2.2: VM migration

A key feature of hypervisors is the ability to migrate VMs, called live migration, which is the process of moving an entire VM from one node to another without interrupting the software running within the VM, as shown in Figure 2.2. Migration enables CPs to move VMs that are not meeting SLAs or conserve energy by consolidating VMs to fewer nodes. There are typically two techniques [76] for VM migration, pre-copy

and post-copy migration. In pre-copy migration, memory pages of the VM are copied from the source to the target node while the VM continues to run. Any memory page changes in the source node are tracked and continuously synced with the destination node. The process continues until either a small number of memory pages or fixed rounds of copying are reached. At that point, the VM is suspended in the source and resumed in the target node. The minimal set of VM states needed to run the VM is copied from the source to the target node in post-copy migration. As the VM runs on the target node, required pages are copied from the source node. The migration time is high in the pre-copy approach, as the pages are kept in sync, and the post-copy approach has a risk of performance degradation to the VM after the VM is resumed as pages are fetched on-demand [135].

The Xen hypervisor [3] is an open-source Type 1 Hypervisor developed by the Xen community. As a bare-metal hypervisor, Xen has its own VM scheduler, which is used to multiplex the execution of the virtual CPUs of the VMs on the physical CPUs on the node [8]. Xen is currently available for the IA-32, x86-64 and ARM binary instruction sets, and can run Linux, Windows, Solaris and BSDs as guest operating systems, and supports Pre-copy and Post-Copy Migration. Xen is under the GNU General Public License (GPL2).

KVM [1] is an open-source hypervisor implemented as a module of the Linux kernel. KVM is a hosted or type 2 hypervisor and runs on a host OS kernel, using the kernel's functionalities. KVM presents guest OSs as regular applications on the hosting OS, and thus they are scheduled by the host OS [8]. KVM's approach reduces the complexity of the hypervisor implementation, as the Linux kernel handles scheduling and memory management.

VMware ESXi [2] is an enterprise bare-metal hypervisor and can accommodate VMs of up to 128 virtual CPUs and 6 TB of RAM. ESXi provides a virtualization layer that abstracts the physical node's CPU, memory, storage, and networking resources and presents these to VMs. ESXi includes an OS kernel that receives resource access requests from VMs and sends them to the physical resources.

Containers, shown in Figure 2.1c, are implemented in the Linux kernel through isolation for one or more Linux processes [150]. Containers are smaller in size and contain less software compared to VMs, and thus they typically are much faster to provision and consume fewer node resources to run. Examples of container engines include Open VZ, Solaris Zones, FreeBSD Jails, LXC, CoreOS (Rocket), and Docker [19]. Similar to VMs, containers can also be migrated due to node maintenance, load

balancing, and node consolidation [110]. Virtuozzo transfers a container's filesystem and virtual memory to a target node and freezes all processes [110]. It then creates a dump of memory state to disk and copies it to the target node; deltas since the dump are also transferred to the target node.

Both hypervisor-based and container-based virtualization have preferred use cases, and neither of the virtualization solutions is best for all applications [145]. Cloud users are likely to use a combination of the two. Similarly, different hypervisors have different energy efficiencies that are dependent on the customer workload [77], and no single hypervisor outperforms the other hypervisors in terms of performance and energy consumption.

## 2.3 Adaptation

To meet workload demands, CPs can apply adaptation [60] to reconfigure resources autonomically. However, there are still some challenges including [11]: granularity of resource adjustments, the startup time of newly provisioned resources and choosing policies for applying elasticity. While a CP's objectives include satisfying customer requests, this may not always be technically feasible or commercially viable. CPs have finite resources and may need to apply prioritisation on requests. Additionally, a CP may decide it is more cost-effective to pay the penalty for an SLA violation instead of scheduling a request. The general view on Elasticity [60] abstracts several complex activities such as combining the decision to adopt and the application of adaptation. We refine this view by separating the decision making process from the reconfiguring cloud environment. We define elasticity as the on-demand ability to scale vertically or horizontally segmented resources in discrete units. To achieve a specific business goal, CPs go through a decision making process that changes the infrastructure, a process we name Cloud Systems Adaptation. We define this as a change to provider revenue, data centre energy consumption, capacity or end-user experience where decision making resulted in a reconfiguration of compute, network or storage resources. Reconfiguration is the process of increasing or reducing resource allocation to a workload through elasticity. Core to cloud systems adaptation is a decision making process that decides the resources to reconfigure. When decision making is complete, elasticity is used to scale the infrastructure resources. For customers to run workloads on cloud environments, these need to be accepted by the CP and mapped onto resources that satisfy both the customer and CP goals. A vital aspect of this process is *VM Placement*, which consists

of two parts: *initial* VM placement, which refers to the first allocation of VMs to nodes in the data centre, and VM *migration* or relocation, which involves the revision of an earlier placement decision. VM placement performs the mapping to meet SLA, energy or profit goal and has been studied extensively [50, 100, 83, 116, 131, 130, 133, 27, 32]; we cover this in more detail in Chapter 3.

In Paper 1 [65], we have shown there are multiple dimensions to cloud adaptations. *Adapting the cloud resource*: CPU, memory, disk bandwidth and storage for VMs, powering on a node or adjusting Dynamic Voltage and Frequency Scaling (DVFS) for a node, adding or removing nodes for cluster adaptations. *Adaptation objectives* are the drivers to engaging adaptation and include reducing SLA violations, reducing power consumption and maximising CP profit. Some of the work in the literature attempts to reduce the cost to the customer [73, 31, 89, 85]. *Adaptation techniques* used on cloud resources include heuristic [167, 59], control theory [12, 161] and machine learning [28, 89]. *Adaptation engagement* is the approach that starts the adaptation process and is typically reactive [167, 136] or proactive [163, 159, 111, 20], with some approaches being a hybrid of both [70, 73]. The *Decision Engine Architecture* governs the method for decision making method, with centralized architectures using a single viewpoint and managing the entire infrastructure. To improve the scalability of centralized architectures, researchers investigated hierarchical and decentralized approaches. Hierarchical architectures aim to enhance scalability and typically divide the infrastructure into multiple clusters [13, 142]. Decentralized approaches distribute the decision making process and have no central control point. These have been shown to achieve good scalability [126, 151, 95]. However, decentralized approaches can suffer in their ability to consolidate VMs due to the lack of a global view [95]. The approaches and scalability of the decision making process are covered in more detail in Chapter 3.

Paper 1 [65] shows that VM level adaptation is typically applied to improve/reduce workload performance due to an increased/decreased demand by adjusting CPU, memory, disk bandwidth and storage. For example, a web server running on a VM may need more CPU share due to an increased number of requests. Node level adaptation could be applied to add capacity by powering on a node. Energy consumption could be reduced by using DVFS before the node is powered off when not needed. Migration can also reduce energy consumption by consolidating VMs into fewer nodes and switching some nodes off. Cluster level adaptation is applied to facilitate node adaptation and adhere to any reliability policies used by CPs by adding and removing

nodes. The included Paper 1 [65] presents an extended analysis of adaptation in cloud computing.

## 2.4 Summary

This chapter introduced some concepts that are building blocks for this research. While virtualization enabled wider adoption of cloud computing, it requires manageability and regular adaptation to balance the needs of CPs and their customers. As the adoption of virtualization increases, there is a greater need for methods to enable efficient management of large scale cloud environments. Through Obj2 and Obj3 we investigate a method for large scale cloud management. Through Obj4 and Ob5 we investigate methods to apply efficient adaptation.

We focus on VM adaptation as this remains a popular virtualization unit and will enable us to compare our proposed approaches to existing work in the literature. It is feasible to extend the proposed methods to manage containers. As there are likely to be more containers than VMs and the proposed approaches in Obj2 and Obj3 aim to solve scalability challenges, we hypothesise the evaluated benefits will also apply when managing containers. The proposed approaches in Obj4 and Ob5 can be extended to capture container metrics, create dynamic threshold levels for container migrations, and manage container migration targets.

# Chapter 3

# Related Work

## 3.1 Cloud Resource Management

This chapter reviews the fundamental semantics of cloud resource management, in particular features of Management Frameworks (MFs) and Management Algorithms (MAs). For MFs, we explore the three common architectures: centralized, hierarchical and decentralized and their key features. We highlight some of the scalability challenges with existing MF approaches and propose an alternative hybrid MF, which can retain the properties of both hierarchical and decentralized architectures and overcome their disadvantages in a public cloud context. The proposed MF models applications as a single VM, and can be extended to manage applications made up of multiple VMs by incorporating a MA that is aware of the constituent VMs that make up an application. We integrate several MAs from the literature with the hybrid MF and carry out an extensive evaluation. To utilise the capabilities of the proposed hybrid MF, we review existing MAs in the literature and develop a new MA that can enable high scalability and cope with the dynamic nature of cloud environments. In the review of MAs, we explore objectives, considered infrastructure resources and allocation techniques. Unique to this review is a focus on the public cloud provider perspective and an introduction of a novel classification of cloud resource management into MAs and MFs.

## 3.2 Management Frameworks - MFs

An MF in cloud resource management provides core capabilities and enables the MA to execute. We extend the identified taxonomy in [124]. Figure 3.1 shows the dimensions in MFs and the relationship of the MA to the MF. Key to the operation of MFs is

the scope of the infrastructure that is made available to the MA, with MF *architectures* typically designed to operate in *centralized, hierarchical and decentralized* manners. The architecture of the MF can influence the *scalability* achieved as the number of the managed resources increases. Additionally, each architecture has different properties for *metric collection* and for providing *reliability*. MFs operate to facilitate key MA functionality, such as placing new workloads on the infrastructure, reacting to existing workloads being stressed or consolidating by optimizing current resources to workload mapping. These operations can be invoked on a *reactive* basis due to workloads or resources, on a *proactive* basis where a prediction is made and executed to achieve a particular goal or a *hybrid* of both approaches.

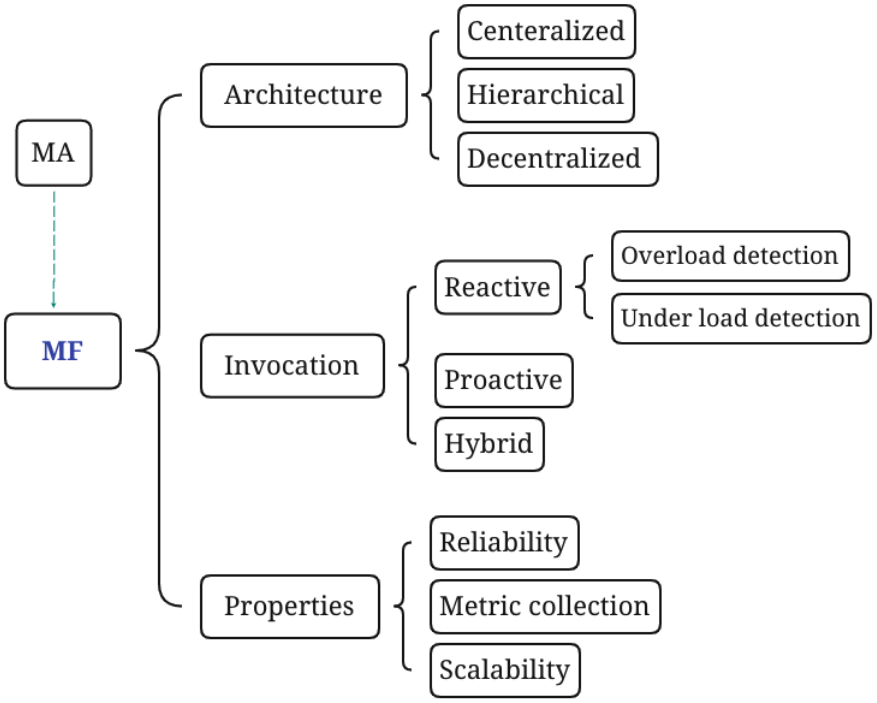



Figure 3.1: MF Taxonomy

### 3.2.1 Architecture

The MF's architecture governs the MA's scope of operation and the infrastructure size the MA attempts to evaluate and adapt during a decision making cycle. A larger scope potentially provides a more extensive search space and thus offers a more optimal optimisation. However, it potentially introduces a challenge with time complexity. In

this section, we examine several MF architectures and the scope they provide to MAs during adaptation.

In centralized MFs, a global controller has a view of all the resources being managed, including physical nodes, VMs or containers and consumption of resources such as memory, CPU, storage and network on each physical node, as shown in Figure 3.2. The MA runs within the global central controller. Thus, the whole data centre infrastructure may be used as input into a single adaptation cycle.

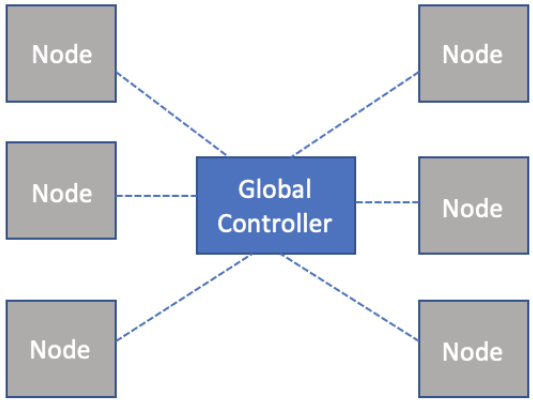



Figure 3.2: Example of Centralized MF architecture

Resource *metric collection* occurs periodically [61], from every node in the entire infrastructure and, typically through the hypervisor interface, with each managed node sending its metrics to the global controller through the network. The collected metrics include CPU, memory, network and storage usage of each VM. These can be input into the adaptation process, and in a centralized architecture, the entire infrastructure is used in adaptation.

Centralized MFs have a single global controller and are the most prevalent in literature proposals [166]. Centralized architectures do not have reliability intrinsic to their design. They require techniques that address issues with a single point of failure [101] to achieve high availability and reliability. Applying redundancy is an essential factor to removing single points of failure, and strategies can be active or passive [45]. There is a master global controller in a passive redundancy, and other similarly capable controllers are on standby. When the master controller failure is detected, a standby controller takes the role of the Master controller and can resume failed node tasks. There are no standby controllers in an active strategy, and all controllers are active and process requests in parallel. If one of the controllers fails, other controllers

can continue incoming requests to manage the infrastructure.

Centralized approaches have been widely utilized in the literature [61, 129, 144, 43, 62]. Entropy [61] is implemented in Xen [39] and has a dedicated node for a global controller. Each managed node has a metric collection capability using Xen's Domain-0. Entropy defines an upper bound for MA execution to manage scalability, and by default, this is set to 1 minute. CloudScale [129] does not assume prior knowledge about the applications running inside the VMs being managed and can apply autoscaling and DVFS adjustment actions. CloudScale is implemented on top of Xen and collects CPU consumption, memory allocation, network traffic, and disk I/O metrics with measurements taken every 1 second. CloudScale only triggers a migration if the violation is predicted to last continuously for K seconds to reduce the number of migrations. K is a configurable value that can be defined and is by default 30 seconds. While Cloudscale was tested on real infrastructure, its ability to scale to manage a large infrastructure was not examined. Google's Borg [144] supports high availability and fault-recovery using a centralized architecture and can manage many thousands of nodes in each cluster, with each cluster receiving 10000 tasks per minute. While Borg can scale to manage a large infrastructure, it has been highly tuned for Google's workload and environment and is suited to long-running services and batch jobs [43]; it is not a generic public cloud MF. Mesos [62] uses a two-level scheduling mechanism, with a centralized master managing slave daemons running on cluster nodes. User frameworks that run tasks execute on the slave nodes. The master process makes resource offers to user frameworks, with the scheduler running each workload deciding to accept or reject the master's offer. Mesos was evaluated to manage 50,000 nodes. However, user frameworks need to be modified to be Mesos aware and thus Mesos is not a generic public cloud MF.

As centralized MFs consider the entire infrastructure during the adaptation and metric collection, they are always subject to scalability, reactivity, and fault-tolerance issues [117]. To address the scalability challenges in centralized MFs, researchers investigated other architectures, including hierarchical MFs. As shown in Figure 3.3, hierarchical MFs typically divide the infrastructure into multiple sections, with a decision engine in each section, which has the effect of creating a distributed centralized MF for each section. In such a section, typically on the cluster level, each node is connected to a cluster manager from which it receives adaptation commands. Within each divided section, metric collection operates in a similar way to centralized MFs.

Hierarchical approaches have been studied in cloud resource management, and the

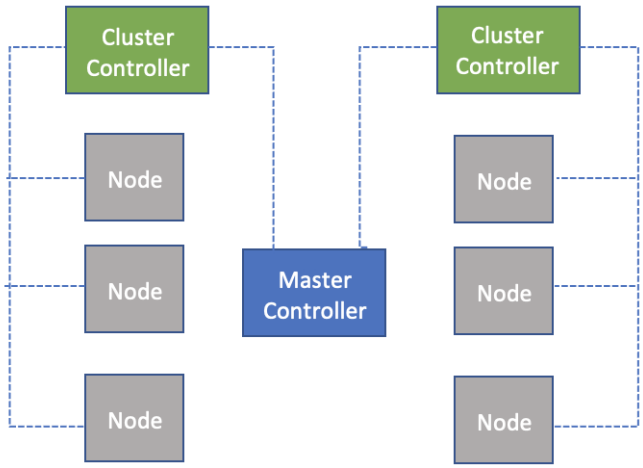



Figure 3.3: Example of Hierarchical MF architecture

approaches typically use a multi-level hierarchical approach running at different time intervals [10, 79, 165, 13, 106, 42, 118]. In Addis et al. [10] the lowest level controller runs every hour and performs VM placement, power management and workload profiling. In [79], the lowest level controllers manage a small number of machines and the applications hosted on them. At the next higher level, a controller manages nodes owned by multiple lower-level controllers. The authors in [165] used three levels, where the highest level controller managed multiple clusters operating at seconds (L1), minutes (L2) and days (L3) intervals. However, the authors did not explore the scalability of their approach. The authors in [13] chose to slice the hierarchy on the operations of the controllers. A Level 1 controller handles VM placement and load balancing and runs every 30 minutes. A Level 2 controller controls the resources of a node and runs every few minutes. The authors in [106] focus on VM placement in their approach using 2 level hierarchical controllers. Some of the proposals considered collaboration among the controllers. In [56], the authors use a hierarchical approach with VM migration escalation and perform the initial assignment of VMs to clusters and periodically, lower controllers decide what to optimise and pass the decision to parent controllers. In [107], the authors outline how a collection of hierarchical autonomic managers can collaborate using messages. Hierarchical proposals typically utilise a controller running in a centralized manner within the scope of a cluster of nodes. Snooze [49] uses a hierarchical architecture to support load balancing and fault tolerance. Each node has a local controller responsible for monitoring the node capacity, including resizing and migrating VMs. A group manager manages a local controller, which acts as an

intermediate layer to propagate metric information and segregate the infrastructure. A group leader manages group managers and has a centralized view of the metrics used to direct VM placement requests. The experiments considered the network load scalability of Snooze but did not evaluate its ability to scale and manage SLAs.

Resource metric collection typically occurs periodically in a similar way to centralized MFs. However, the metrics being sent to a global controller are sent to a controller managing a smaller subset of nodes at the rack or cluster level [79]. As the number of nodes within the cluster increases, the decision making algorithm faces a similar challenge to traditional centralized approaches. The execution time can result in an inability to react to SLA violations.

Hierarchical approaches improve the reliability properties compared to centralized approaches, as they divide the managed infrastructure and increase the number of managing controllers. However, hierarchical approaches suffer similar challenges to centralized, at the single cluster level, and require explicit redundancy strategies. For example, in [10] each cluster maintains primary and backup controllers.

While there have been attempts to examine the scalability of hierarchical approaches [10, 56], these tend to be small or do not examine an increasing size of the managed infrastructure.

Cloud resource management solutions need to maintain VM placement that meets SLA agreements and optimise the usage of CP resources. The scalability of the MF has a significant impact on the ability of cloud resource management solutions to respond to SLA violations [90]. Decentralized architectures distribute the management of the infrastructure without a centralized controller [118], and both hierarchical and decentralized architectures are a form of distributed management architectures. A typical layout for a decentralized MF is shown in Figure 3.4.

Some of the decentralized approaches use an overlay network as a mechanism for nodes to communicate. An overlay is a logical network that runs independently of a physical network and does not change the underlying network. Examples of overlay networks include Peer-to-peer (P2P), virtual private networks (VPNs), and voice over IP (VoIP) [141].

DVMS [117] has a decentralized MF, where an agent runs on each node to manage the VMs running on the node and collaborate with other neighbouring agents. Agents communication is done through a fault-tolerant overlay network that relies on a distributed hash table. The communication overlay defines the node to neighbour relation

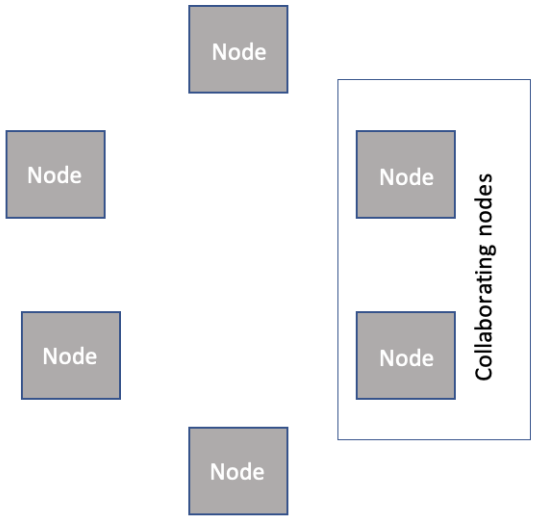



Figure 3.4: Example decentralized MF architecture

and can be structured or unstructured. Each node can submit new workloads and handle cooperation events to manage the infrastructure generated by other nodes. Each node monitors its resource usage and can create a partial view of the resource usage on neighbouring nodes, on-demand. The partial view improves scalability and enhances fault tolerance as nodes can communicate with new neighbours in the event of failure of some nodes. Metric collection is performed by a resource monitor, which updates monitoring information every two seconds. If a violation is detected, an event is generated to resolve the stress by collecting neighbouring metric details. The authors in [152] proposed a decentralized approach that uses a gossip protocol to enable node communication. The protocol has a distributed algorithm, where each node regularly selects a subset of other nodes to interact with using small-sized messages. The protocol uses a push-pull paradigm, where two nodes exchange state information and update their local states during each exchange. The approach appears to be simple, scalable and more robust. The protocol runs on each node and utilises metrics on available data centre resources and the current resource demand. Each node has a partial view of all the nodes at any point in time. Simulator evaluations show the approach can scale to many thousands of nodes. However, the approach operates with a partial view of the infrastructure, and a node only cooperates with another randomly selected node. It may take several interactions with multiple nodes before a node can find a target node within the overlay to migrate a VM during a stressful situation. The authors in [114] proposed a decentralized approach to manage an extensive cloud infrastructure, where

Table 3.1: Summary of MF Architectures

| | **Centralized** | **Hierarchical** | **Decentralized** |
|---|---|---|---|
| *Operation scope* | Global | Partial | Partial |
| *Scalable by design* | No | Partial | Yes |
| *Built-in reliability* | No | No | Yes |
| *Metric collection* | All nodes to master controller | Cluster nodes to cluster controller | One node to cooperating nodes |

nodes are organised in an overlay network. Nodes can operate autonomously and can scale up and down in response to changes to workloads. Underloaded nodes cooperate to move their workload to other nodes and switch off, while overloaded nodes can migrate VMs to reduce energy consumption and reduce SLA violations. The protocols for managing the overlay network have a time complexity of $O(\log_2 N)$, where N is the number of nodes being managed.

Table 3.1 summarises properties of the centralized, hierarchical and decentralized architectures and shows that decentralized approaches typically have inherent scalability but can be limited by their partial view of the infrastructure.

### 3.2.2 Invocation

The cloud adaptation process can be expensive to carry out decision making and to execute state adaptation [158], thus there is a need to balance the benefit and the running cost of adaptation. This raises the challenge of deciding when the adaptation process should be invoked to achieve this balance. This section explores the current approaches and challenges in invoking adaptation and summarises our contribution of a dynamic method to invoke adaptation.

The methods used in the literature fall onto reactive, proactive or hybrid engagement. Reactive approaches invoke adaptation when a monitored metric, e.g. CPU utilisation, reaches a specific threshold or when an event is received, such as a new VM placement or termination request. Proactive approaches predict what demands will be placed on the infrastructure and invoke adaptation ahead of the expected resource contention point. Hybrid approaches utilise proactive methods and combine these with reactive methods to engage adaptation for long and short term time scales.

Examples of approaches that predict invocation include [79, 13, 129]. In Mistral [79], a workload predictor estimates the stability interval following the current time. For an application, Mistral predicts the period when the workload remains within a

band. At each monitoring period, Mistral checks if the application is within its band, and re-estimates the next stability interval for applications outside of their predicated band using an autoregressive moving-average technique. The controller chooses which (if any) of the available adaptation actions to take to improve the overall utility of the managed system. In [13], a predictor forecasts system workload based on previous monitoring data. A resource allocator then uses these to calculate the capacity required by VMs. The prediction of future utilization is based on a moving average of utilisation in the previous $n$ intervals. In CloudScale [129], resource usage time series are used as input into a resource demand prediction model, which predicts short term demands. The model uses a fast fourier transform to identify any repeating patterns, and if found, it is used to estimate future resource demands. If not found, the model uses a discrete-time Markov chain to predict resource demand.

Examples of approaches that adapt reactively include [117, 49, 10]. The authors in [117] use a reactive approach, with adaptation being invoked when an event is received. Events can include new VM placement or stress requests generated by self-monitoring nodes. The authors argue this is better than a predictive approach that requires accurate knowledge of workload profiles. Nodes self-monitor using CPU and memory thresholds set by CPs and by default collect metrics every two seconds. In Snooze [49], a reactive approach is used to execute policies, such as scheduling, optimising and planning. For example, an administrator can configure optimization to be run during a specific hour in the day. In Addis, [10] the lowest level controller runs every hour and performs VM placement, power management and workload profiling.

Overload detection determines if the managed node is under stress and may not fulfil its SLA obligations and is typically applied to node CPU utilization. Once an overload state is detected, an adaptation process may be invoked to relieve a stressed node through VM selection and migration from the stressed node to a chosen target node. Conversely, underload detection determines if the managed node is underutilized and may invoke a consolidation process to migrate VMs from nodes with low CPU utilisation to nodes with higher CPU utilisation and looks to power off underutilized nodes to reduce energy consumption [128].

Reactive approaches are typically implemented using threshold techniques [41] by triggering adaptation when a node's utilization reaches a given level. Beloglazov et al [25] proposed a collection of adaptive policies for setting the upper thresholds: Interquartile Range, Median Absolute Deviation, Local Regression, and Robust Local Regression. The thresholds are calculated through statistical analysis of historical

node utilisation metrics. Other approaches include adaptive heuristic algorithms [155]. The authors in [102] proposed an overload and underloaded node detection. A node is deemed overloaded if the actual and the predicted total CPU usage of 7-time intervals ahead exceed the defined overload threshold. Different to [25], the authors additionally incorporated a probabilistic approach to counter the uncertainty of the long-term predictions as well as the cost of applying the VM migration. Other proposals include a regression-based algorithm to create an upper threshold for detecting overload [155]. The approach automatically adjusts the upper CPU utilization threshold based on the historical CPU utilization of the nodes.

A key element to the threshold-based approach is the assumption there is a high chance that an overload occurs when a node's utilization exceeds the set threshold. Thus, the threshold level creates an association between a node metric, eg CPU utilization, and SLA violation. However, the metric threshold where SLA violations can occur varies based on the application and the node configuration. Creating a single threshold for all application and node configurations is incredibly difficult. While the current approaches can reduce overload, they can limit the utilization gains that can be achieved as they leave unused slack for each node. Additionally, threshold approaches can trigger unnecessary migrations as exceeding the set threshold does not necessarily equate to an SLA violation [41]. Given these challenges, we identify invocation of adaptation as an area of further research, and we propose a method to dynamically learn a CPU utilization threshold based on a derived reward, in Section 4.5. In this approach, each node performs exploratory VM migrations at different CPU utilisation levels and determines a reward for each level, converging on a threshold that balances SLA violations and maximises CPU utilization.

### 3.2.3 Discussion - MFs

Table 3.2 shows MFs from the literature and how they invoke the adaptation process, their architecture and the type of resource management operation they perform. Our review shows these MFs are mainly reactive and all perform VM placement, with some performing adaptation to stress and consolidation of VMs onto fewer nodes.

Centralized, hierarchical and decentralized architectures have their merits. Centralized architectures have a global view, hierarchical architectures have increased scalability compared to centralized ones, and decentralized architectures have no central controller. They can scale to manage a large number of nodes. Centralized architectures are common in the literature and have a more straightforward design and

implementation than hierarchical and decentralized MFs, due to their command control mechanism. Hierarchical architectures share some of the simplicity of the centralized approach but with the added complexity of needing to propagate metrics to higher-level controllers. While decentralized architectures can achieve large scale, they have higher complexity in their design due to the close cooperation between nodes to achieve resource management objectives [40]. Additionally, in the current decentralized approaches, each node cooperates with other nodes in its overlay with no ability to migrate a VM outside the overlay. In a stress situation, this could result in a node remaining in a stressed state for an extended period until a suitable migration is found inside the overlay. Our hypothesis is the strengths of these architectures can be combined, and their weaknesses overcome. We investigate a hybrid MF that overcomes the shortcomings in hierarchical and decentralized approaches and reduces SLA violation metrics. We present this approach in contribution 2, Paper 2 [66], in Chapter 4, where we show the proposed hybrid MF can scale to manage an extensive infrastructure and reduce SLA violations.

Table 3.2: Comparison of MF architectures

| **MF** | **Architecture** | **Invocation** | **Operation** | | |
|---|---|---|---|---|---|
| | | | ***Placement*** | ***Stress*** | ***Consolidation*** |
| Addis [10] | Hierarchical | Predictive | x | x | |
| Jung [79] | Hierarchical | Predictive | x | x | x |
| Almeida [13] | Hierarchical | Predictive | x | | |
| Hindman [62] | Hierarchical | Reactive | x | | |
| Quesnel [117] | Decentralized | Reactive | x | x | x |
| Pantazoglou [114] | Decentralized | Reactive | x | x | x |
| Wuhib [152] | Decentralized | Reactive | x | x | |
| Hermenier [61] | Centralized | Reactive | x | | x |
| Shen [129] | Centralized | Predictive | x | x | |
| Verma [144] | Centralized | Reactive | x | | |

To assess the extendibility of the proposed hybrid MF, we implement multiple MAs from the literature, evaluate the quality of service metrics of the hybrid MF, and compare these to centralized, hierarchical, and decentralized architectures. In the hybrid MF, nodes are autonomous, decide when to accept migration requests and are typically less stressed than nodes in the other evaluated architectures. This results in lower VM migration instability and enables more opportunities for VM consolidations. We have

shown these factors lead to lower SLA violations and less migration traffic. Additionally, the escalation approach in the hybrid MF attempts to service resource requests at the lowest local level possible to reduce the overhead of servicing the request. We demonstrate the hybrid MF reduces SLA violations compared to centralized, hierarchical and decentralized architectures, particularly in high-stress workloads. We present this in contribution 3, Paper 3 [64], in Chapter 4.

The review of MFs in the literature highlighted some of the challenges in *invoking* adaptation, particularly with the current threshold-based approaches. There is an opportunity to investigate a dynamic approach to setting a metric threshold, such as CPU utilisation, which considers the impact of the set threshold level. For example, one option is to perform online learning of a threshold that gives a node utility for a given CPU utilisation. This would enable the threshold to factor in the dynamic nature of cloud environments and their heterogeneous workloads. As part of contribution 5, we investigate and develop a method that enables a node to dynamically learn a CPU utilization threshold based on a derived reward. In this approach, each node performs experimental VM migrations periodically at varying CPU utilisation levels and calculates a reward for each of the used threshold levels, converging on a threshold that balances SLA violations and maximises CPU utilization. We present this approach in contribution 5, Paper 5 [7], in Chapter 4.

## 3.3 Management Algorithms - MAs

The MA is the decision making component that assigns workloads to infrastructure resources by regularly assessing the satisfaction of such assignments in achieving a given objective, which typically includes SLAs. The frequency and running time of the MA have a significant impact on its ability to map workloads to infrastructure resources efficiently. The lower the complexity, the more frequently the algorithm can be executed.

Given the dynamic properties of cloud environments and the regular change in the structure of workloads, our hypothesis is a dynamic MA that can regularly learn will be well suited to balancing the goals of meeting SLAs and conserving energy consumption. Our goal is to investigate a MA that can be combined with the proposed hybrid MF, discussed in the last section of this chapter. The hybrid MF has been shown to scale and manage a large infrastructure [66] by combining a decentralized approach with the ability to escalate through a hierarchical architecture. This section reviews

MA proposals in the literature and identifies an MA to use with the hybrid MF.

The proposed novel MA classification, shown in Figure 3.5, has three key aspects; *Objectives* define the goal of the adaptation process and typically includes minimising SLA violations, by using the MFs invocation capability to detect overloaded node state and carry out stress resolution. Objectives also include reducing energy consumption by using the MFs invocation capability to detect underload and migrate to consolidate VMs. Other energy minimising approaches include Dynamic Voltage Frequency Scaling to reduce the speed and power consumption of processors in nodes. The *allocation techniques* are a set of analytical and modelling techniques used to achieve the adaptation objective and include heuristics, metaheuristics, control theory and machine learning. The *considered resources* are infrastructure components used either to monitor the state or are reconfigured as part of the adaptation process.

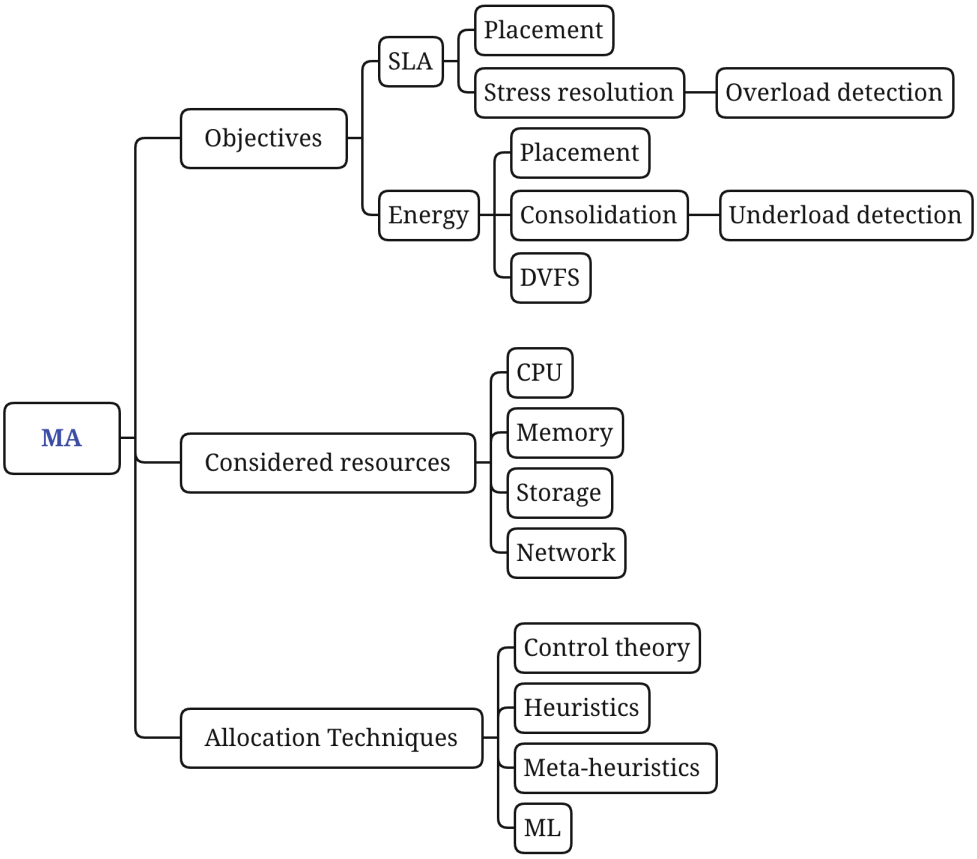



Figure 3.5: MA Taxonomy

### 3.3.1 Objectives

The objective of a VM Placement is to map customer workloads onto Cloud Providers (CPs) resources in a way that achieves a particular objective, such as reducing energy consumption or load balance while ensuring SLAs are met [90]. VM placement consists of two parts: *initial* VM placement, which refers to the first allocation of VMs to nodes in the data centre, and VM *migration* or relocation, which involves the revision of an earlier placement decision. VM placement performs the mapping in order to meet SLA, energy or profit goals and has been studied extensively [50, 100, 83, 116, 131, 130, 133, 27, 32].

#### SLA

A Service Level Agreement (SLA) defines the quality of service (QoS) requirements as a contractual expectation between the customer and CP. Achieving SLAs is part of a CPs operating model and is a crucial objective for adaptation. Components of SLAs include *service guarantees*, which are the metrics the CP aims to meet, and have: availability and workload response time [23]. *Service granularity* calculates an aggregate of the contracted resources. For example, aggregate uptime of specific resources is contracted at a certain percentage. As these are typically aggregates, it implies that some periods may be lower than the specified level [23]. An SLA violation is when the CP does not meet the defined SLA. Many CPs leave the responsibility for reporting SLA violations to the customer [23]. Given the complexity of proving SLA violations for cloud customers, some of the literature aims to help customers monitor SLAs and detect violations by implementing advanced SLA management strategies and considering the semantic meaning of SLA concepts [88]. Many of the approaches aim to focus on minimising SLA violations, with many reducing the number of VM migrations.

As the CPs business model depends on meeting SLAs, many of the approaches shown in Table 3.3 aim to meet customer SLAs. We identify this as a critical property to achieve in the MA to pair with the hybrid MF.

#### Energy consumption

In cloud environments, optimizing the infrastructure's energy consumption is a significant challenge for CPs from both environmental and economic perspectives [37].

Inefficiency in VM placement strategy could significantly impact the quality of service, and the amount of energy consumed in a cloud data centre [18]. Thus improving energy consumption in data centres involves improving the entire stack of components and operations, including energy usage of physical nodes, storage and network equipment, dynamic voltage and frequency scaling (DVFS), and consolidation of VMs onto fewer nodes and switching unused nodes off [17]. The average CPU utilisation of nodes within data centres is between 15%–20% [63] with the idle state being the typical case. The consolidation process migrates VMs from nodes with low CPU utilisation to nodes with higher CPU utilisation and looks to power off underutilized nodes to minimise energy consumption [128].

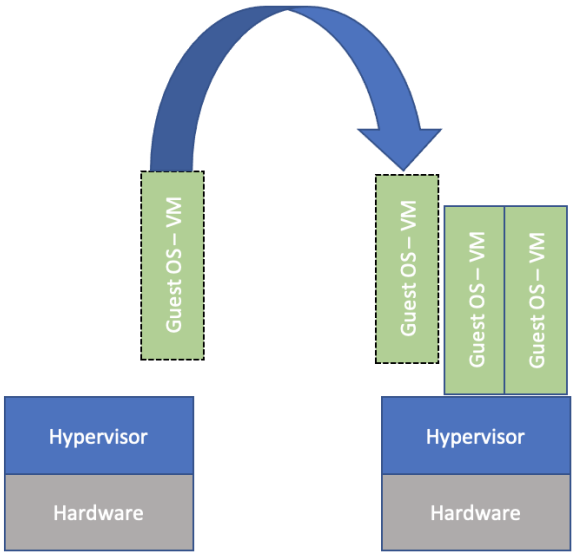



Figure 3.6: Consolidation process

VM consolidation to optimise energy consumption has been extensively investigated. Consolidation aims to reduce the number of active nodes within the infrastructure by revising the VM to node mapping and optimising for a reduced number of running nodes, shown in Figure 3.6. However, consolidating a large number of VMs on a highly utilized node can lead to SLA violations. Thus, the consolidation process needs to balance conserving energy with SLA violations. Equally, to achieve its goal, the consolidation process needs to avoid placing a VM on an underutilized node [81]. The authors in [58] propose an approach to increase the efficiency of node utilization on active nodes across the data centre. The authors in [123] propose a consolidation approach that aims to reduce energy consumption and complete more tasks inside VMs. The approach uses workload utilization across the data centre to set lower thresholds.

Dynamic voltage frequency scaling (DVFS) is a commonly used power management technique that targets energy dissipation [37] by dynamically scaling the frequency and voltage of the microprocessor at runtime. The authors in [103] proposed a heuristic for selecting a VM for each task to optimize the energy utilization by applying the DVFS technique. The approach sets VMs to satisfy the task SLA even with reduced processor performance, achieved through DVFS. The authors in [16] classify consolidation into multiple sub-problems and propose a DVFS-aware consolidation. The authors in [103] proposed a heuristic for the selection of a VM for each task to optimize energy utilization by applying the DVFS technique. The approach selects VMs that can satisfy the task SLA even with reduced processor performance, achieved through DVFS. The authors in [16] classify consolidation into multiple sub-problems and propose a DVFS-aware consolidation.

Table 3.3 shows that conserving energy consumption is widely researched. However, there are still challenges with identifying the benefit versus migration cost during consolidation. We identify this as additional research that could be improved with a dynamic MA that can learn a policy from the current environmental conditions and decide on the benefits of applying consolidation and DVFS. This is an area of research, which is outside the scope of this thesis.

### 3.3.2 Considered Resource

An adaption process can reconfigure cloud resources to achieve a particular objective. Examples of considered resources are summarised as follows:

- CPU & Memory: they are core computing resources, and proposals consider metrics that scale these horizontally by adding new VMs, typically via predefined VM classes. Alternatively, fine-grain adjustments have been achieved through the Hypervisor API interface [39]. Approaches in the literature consider both physical and virtual CPU resources, and in this survey, we consolidate both approaches under CPU. Dynamic voltage and frequency scaling (DVFS) can also be applied to the CPU to conserve energy.

- Disk: Storage disk adaptation can be applied for scaling the size or to determine the location where the disk resides to enable VMs virtual disks to be spread out across the physical disks of a node [6].

- Network: several approaches aim to use network bandwidth as a metric for VM

allocation. Some focus on optimising network utilization to reduce costs for a CP customer [6].

### 3.3.3 Allocation Techniques

In this section, we review different techniques that have been used to implement MAs in the literature. Our hypothesis is MAs that can learn online are suited to the dynamic workloads exhibited in public clouds, and these MAs can be paired with the proposed hybrid MF and further utilise its properties.

Several allocation techniques have been applied to cloud infrastructure in the literature, including heuristics, metaheuristics, control theory and machine learning. These are presented next.

#### Heuristic & Meta-heuristic

Heuristic approaches are widely used in the literature and include: *Random PM Selection* [81] where a target node is chosen randomly during VM migration. In *First Fit*, a list of possible targets in the search space is ordered, and each migrating VM is assessed to fit on a node in sequence until a matching node is found. *First Fit Decreasing* is similar to First Fit, with the addition of VMs being sorted in decreasing order of resource usage. Thus, the search starts with a VM with the highest resource demand. *Next Fit* is similar to First Fit, except the search starts from the last stop point during the MAs operation. In *Best Fit,* the node with the minimum residual resource is selected as a target node. The residual resource is the delta of node capacity and the aggregated resource demand of all hosted VMs on the node, including the new VM being migrated. *Best Fit Decreasing* is similar to Best Fit, except VMs are sorted in the decreasing order of their resource demand. Thus the search starts with the VM with the highest resource demand.

Other heuristic approaches include [58], where the authors propose a method to increase the efficiency of node utilization and balance utilization of CPU and memory usage on active nodes across the data centre. The approach uses a multi-dimensional resource usage model for target node selection to guide the VM placement process. A resource usage factor is assigned to each node and used in node selection. Their experiments show minimisation of low utilized resources and more balanced utilization of CPU and memory usage on active nodes. The authors in [123] propose an approach that aims to reduce energy consumption and complete more tasks inside VMs.

They suggest a performance-to-power ratio to set the upper thresholds used in overload detection. For consolidation, the approach uses all workload utilization to set lower thresholds. The authors in [153] proposed a multi-constraint optimization model by considering migration cost and remaining VM migration runtime and using a heuristic policy. The applied constraints were that the total CPU/memory requirements allocated to the VMs should not exceed the node's resource capacity. A VM should be assigned to a single physical node; a node's maximum duration in SLA violation and the remaining runtime for a VM was also considered.

While heuristics are common in the literature, they are typically problem-based strategies that guide the search in improving the current solutions to the problem. However, heuristics usually do not find a global optimum and may provide locally optimal outcomes [166, 109].

*Metaheuristic* strategies are another technique used for cloud resource management and include Genetic Algorithm, Particle Swarm Optimization and Ant Colony Optimization. In contrast to heuristics, metaheuristics can provide a global optimal point, although these techniques typically have a higher time complexity compared to heuristics [166]. The authors in [44] present a review of multi-objective VM placement mechanisms using nature-inspired metaheuristic MAs. They classified the multi-objective literature into population-based, Single solution-based and Hybrid. The authors in [125] proposed a framework for controlling energy consumption and achieving customer's satisfaction. The approach uses multiple phases with a single-objective optimization performed using a genetic algorithm to optimise energy consumption and task completion. Multi-objective optimization is then carried out using a genetic algorithm to optimise both energy consumption and task completion. Additionally, the approach uses an artificial neural network to predict the available resources based on their features and the characteristics of the VMs. The authors in [14] propose a hybrid multi-objective MA based on the swarm and sine-cosine algorithm, which optimises the meantime before a node shutdown, energy consumption, and SLA violations. The time complexity of the MA depended on several factors, including the dimension of the tested problem, the number of solutions, the number of objectives and the maximum number of iterations. The authors in [9] propose a hybrid MA based on a genetic algorithm and multidimensional resource-aware best fit allocation strategy. Their MA aims to improve the energy consumption rate by minimizing the number of active nodes and the usage rate of node resources. The approach takes n virtual machines and m physical nodes, and the MA suggests different permutations for mapping VMs to nodes to

minimise energy consumption. The authors in [57] propose a hybrid multi-objective optimization, using a genetic algorithm and mixed-integer Linear Programming approach. Their results show the complexity of the decision making is further increased by the modelling of Dynamic Voltage and Frequency Scaling (DVFS).

Many of the approaches tackling multi-objective optimizations use a metaheuristic, however, the authors in [139] propose HUNTER, a resource management technique that formulates the goal of optimizing energy efficiency in data centres considering the combination of energy, thermal and cooling.

### Control Theory

The implementation of cloud adaptation using *control theory* techniques typically uses a feedback loop, where a controller maintains the output of the controlled system towards a given target. The controller periodically monitors the inputs and results, for example, the throughput of a system [140]. The authors in [113] used a two-layered controller using classical control theory and focused on multi-tiered applications running on the same node. An adaptive integral controller runs on a VM and maintains the target CPU utilisation for a VM by adjusting the CPU allocation. A higher-level controller is responsible for allocating the CPU share based on the requested CPU allocations by each VMs utilisation. The approach in [146] predicts the performance of applications running on VMs and proactively adjusts VM resources to meet SLAs. An application controller is responsible for calculating the required CPU to achieve SLAs, and a node controller manages the assigned CPU share for each VM. A key challenge with control theory approaches is the complexity of choosing the model gain parameters, potentially leading to system instability [108].

### Machine Learning

The MA techniques covered so far have shown promise in optimizing resource usage. However, cloud environments are highly complex and are typically multi-tenanted with non-linear workloads; as a result, they experience high variability. Machine Learning (ML) techniques can offer an opportunity to adjust resource management in a dynamic way, which is reflective of the context of cloud environments [82]. ML techniques include *Supervised Learning* where every data sample is labelled and used as input. The learning process works by associating features of the input and human feedback. In *Unsupervised Learning* samples are used as input, but unlike supervised learning, there are no labels, and the learning process aims to learn the data distribution within

the sample. For example, VM usage patterns can be used to cluster VMs into distinct groups through unsupervised learning. In *Reinforcement Learning* there is no labelled input. Instead, an agent learns dynamically from its environment and balances the exploration of new knowledge versus the exploitation of known knowledge.

Some ML approaches focus on auto-scaling resources, autonomously provisioning and de-provisioning resources. The authors in [108] presented an auto-scaling method for adaptive provisioning of elastic cloud services, based on ML time-series forecasting and queuing theory, aimed at optimizing response time. The approach uses Support Vector Machines (SVM) to predict the average node load for the following hour and then a queuing model to adjust the resources assigned to a node. Their experiments show SVM has better prediction than moving average and linear regression. Similarly, another prediction approach was presented in [154] with Long Short Term Memory (LSTM) time-series prediction and provisioning through queuing theory. Their results show LSTM performed better prediction accuracy than SVM and autoregressive integrated moving average. A neural network technique was presented in [149], which proposes an adaptive selection that can choose a VM consolidation approach based on the current environment and the cloud provider's priority on energy and SLA violation. The approach firstly generates a raw dataset by simulating the methods for several time steps. Each row will contain the initial environment parameters and normalized evaluation results of all policies. The results (energy and SLA violations) for each row are then normalized. A performance score is calculated using the evaluation priority and normalized evaluation result from the raw dataset; this score is then used to train the neural network. Another framework for resource reservation is presented in [132], based on load prediction and several ML approaches, including neural networks and linear regression. The approach takes an initial reservation plan and monitoring data as inputs and optimises the plan based on observations, with the CPU being the primary monitored resource. The evaluation shows a neural network yielded better predictions.

While ML models approaches are effective, one limitation of these approaches is that they are typically trained offline and require retraining to use new data [128]. Cloud environments are dynamic and exhibit regular changes in the structure of workloads and access patterns. Aptly, RL can operate online, learn dynamically from interacting with a changing environment, and use new information to enhance decision making. RL approaches do not require prior knowledge of the optimization model and are not coded explicit instructions relating to which action to take next; instead, they learn actions through feedback from the environment. These features make RL well

suited to cloud resource management [108]. RL is a machine learning technique that enables an agent to learn within an interactive environment through trial and error and uses signals from the environment in a feedback loop. In Figure 3.7, the agent monitors the current state of the environment (Step 1) and chooses an action from the available options on the environment (Step 2). The environment will then generate a reward for the action taken by the agent and transition to a new state (Step 3). The goal-oriented agent aims to learn the set of actions, a policy, that will lead it to a specific goal or maximise an objective function. RL problems are typically formulated with well-defined transition probabilities and modelled as a Markov Decision Process (MDP) [134].

RL approaches are categorised as model-based or model-free methods, depending on whether full model knowledge can be specified. Model-based approaches need knowledge of the environment model, while model-free strategies learn a policy based on observations and rewards [134].

There are two common control categories of RL. *Value-based or off-policy methods*: RL algorithms proceed to learn the value function for every state/state-action pair to arrive at the optimal policy. Q-learning is a standard algorithm in this category. *Policy-based or on-line methods* directly learn the parameters for the policy, instead of learning an explicit policy function, by fine-tuning a vector of parameters to select the best action to take for policy. SARSA, State–action–reward–state–action, is a typical example in this category [134].

Deep reinforcement learning combines RL with a deep neural network-based approximation of expected values. An offline phase prepares the network with prior system knowledge, for example, from execution. These are then used during online RL execution to select the best actions based on the state of the environment [94].

Q-learning [147] is a common control strategy in cloud resource management due to its simple implementation. Q-learning is an RL algorithm belonging to a collection of algorithms known as Temporal Difference (TD) methods. Q-learning estimates the optimal action-value function, independent of the followed policy, and does not require a complete environment model. The action-value function or Q-function is updated using Equation 3.1 and approximates the value of selecting a specific action at a particular state.

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma MaxQ((s_{t+1}, a_{t_{+1}}) - Q(s_t,\ a_t))] \tag{3.1}$$

In this equation, $\alpha \in [0,1]$ is the learning rate, or step size, and determines how the agent learns from recent updates. A high value for $\alpha$ means the most recent information is

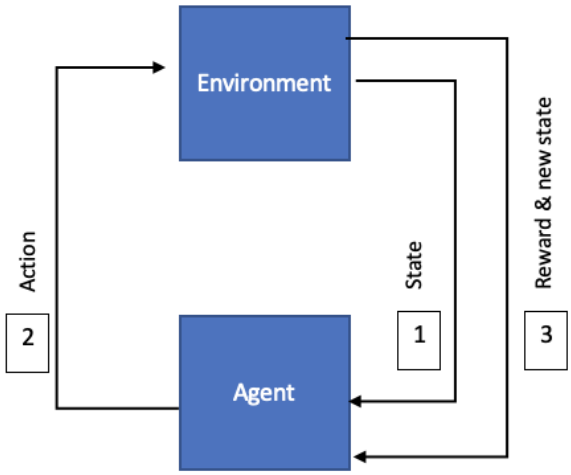



Figure 3.7: RL continuous process

utilized, while a low value implies slower learning. To dampen the reward's effect on the agent's choice of action, the discount factor $\gamma \in [0,1]$ is used. When $\gamma$ is set to 1, the agent will emphasise greater weight to rewards in the future. When it is closer to 0, the agent will only consider the most recent rewards. $MaxQ(s_{t+1}, a_{t_{+1}})$ returns the maximum estimate for the future state-action pair. Once the Q-Value is calculated, it is then stored in the agent's Q-Table.

Auto-scaling of the assigned VMs is the focus of some of the approaches in the literature by using RL to add more resources for customer workloads. The authors in [78] propose a general-purpose model-free learning algorithm based on Q-learning that adapts to unknown system specifics, such as application traffic, to generate scaling up or down actions. We propose a Q-learning based MA and focus on migrating stressed VMs as opposed to auto-scaling, in Section 4.4.

To speed up the convergence of RL, the authors in [22] develop an approach that parallelises Q-learning to speed up the convergence of agents to achieve auto-scaling of VMs. We propose a similar method of parallel learning and further enhance it with co-operative learning between agents running at different layers of the hybrid hierarchical decentralized MF [67].

Reinforcement learning techniques can suffer from the curse of dimensionality. The state and action space can grow exponentially, introducing challenges in the time needed for the RL agent to explore a given environment. To address this, some approaches utilise function approximation, such as Deep Q-Learning (DQN) [105], which

combines deep learning and Q-learning to combat the challenges of Q-learning in environments with a large or continuous state-action space. The authors in [80] propose a DQN based model to respond to anomalies in CPU and memory bottlenecks and apply granular actions to autoscale resources. The approach in [30] also proposes a Deep Q-learning based approach to adjust the size of a cluster by taking the state of the cluster as input and training an RL agent to resize a cluster based on administrator-defined policies and rewards. The agent can use Deep Q-learning, Double Deep Q-learning or Full Deep Q-learning, and the approach was compared to other RL and decision-tree based approaches and shows it gains rewards up to 1.6 times better. Alternatively, to using a DQN, the approach in [120] used a coarse-grained Q-table to achieve higher resolution in the Q-table with less cost. The approach proposed granular actions to adjust CPU and memory resources and applied them in Q-learning's distributed learning mechanism. The work in [35] used a heuristic method to reduce the state space to a smaller set by dividing the original state space into multiple exclusive subsets, where a range of states can fit into the same subset, thus reducing the state space to aid RL convergence speed. Other non-statistical approaches for function approximation have been proposed [74, 15, 29]. Instead of function approximation, we use an aggregation approach in our proposed MA to address the curse of dimensionality. This reduces states and actions into smaller groups, with multiple states being mapped into a smaller number of states and actions.

Public cloud platforms are capable of optimizing the performance of VMs by adjusting the resources allocated to VMs [156]. However, cloud environments are dynamic and undergo regular change, which raises an opportunity for dynamic decision making to optimise resource mapping and cope with the uncertainty of cloud environments. RL is increasingly being used in cloud resource management to address challenges of constantly changing environments

*Scaling* of the assigned VMs is the focus of many of the approaches in the literature by changing the number of VMs assigned to an application. The authors in [78] proposed general-purpose model-free learning algorithms, based on Q-learning, that adapt to unknown system specifics such as application traffic to generate scaling up or down actions.

*Migration* of stressed VMs that are failing quality of service metric is the focus of some of the work in the literature. The authors in [122] propose an approach that uses a DQN technique and a neural network, long-term and short-term Memory, for function approximation. The architecture of the approach is split into offline and online

components. The offline part trains the learning agent by sampling log data generated by the online agent. The online agent has a similar method to the offline, except it does not update the agent parameters, and online decision information is used for the next offline training. The authors in [112] propose a Q-learning controller that responds to volatile and complex arrival patterns through a set of simple states and actions. The controller is implemented within a decentralized architecture, with each node responsible for maintaining its SLA performance. The approach can scale up and scale down by shutting down excess nodes to save on energy consumption. To combat the state space challenge in Q-learning, the approach uses a reduced state space. To determine the state of an application, a linear regression method of response time is used. The authors in [157] uniquely investigate VM migration during data centre upgrades and use a DQN to decide the destination nodes machine for each VM migration to minimise the total migration time. The approach does not require prior knowledge of the data centre topology.

This thesis develops an RL-based controller to solve the VM migration problem and combines Q-Learning with an aggregated state action space to address the state/action complexity in Q-learning. To speed up RL convergence, we utilise parallel learning agents that learn from a shared collective experience of all agents. We develop a reward function that focuses on learning a policy to reduce SLA violations and balance energy consumption [67].

### 3.3.4 Discussion - MAs

A key element of cloud resource management is the overall scalability of the approach. While much of the focus in the literature is on MAs, this thesis is grounded on the hypothesis that solving the scalability challenge starts with addressing scalability in the MF. To address this, we propose a novel scalable hybrid MF and aim to pair it with a MA that can utilise the features of the hybrid MF. In this subsection, we draw some conclusions from reviewing MAs in the literature and relate these to the research objectives in this thesis.

VM migration is a valuable feature that enables adaptation and helps CPs achieve SLAs and performance and reduce energy consumption. However, VM migration can cost both the source and target nodes of the migration, about 10% CPU overhead [158]. While some of the approaches in the literature aim to minimize the migration cost [153, 123, 58], this should be balanced with the derived benefits of performing the VM migration. An additional drawback of the current VM migrations approaches is

Table 3.3: MA Taxonomy
Key: MIP: Mixed integer programming, ACO: Ant Colony, SS: Single Solution, GA: Genetic Algorithm, WOA: Whale Optimization Algorithm, SSA: Salp swarm algorithm, SCA: Sine-cosine algorithms, SVM: Support Vector Machines, QT: Queueing theory, Q: Q-learning, DQN: Deep Q Learning, SARSA: State–action–reward–state–action, ML: Machine learning, CT: Control theory.

| **MA** | **Objective** | **Considered Resource** | | | | **Techniques** |
|---|---|---|---|---|---|---|
| | | ***CPU*** | ***Mem*** | ***Storage*** | ***Network*** | |
| Yadav[155] | SLA & Energy | x | | | | Heuristic |
| Minarolli[102] | SLA & Energy | x | | | | Heuristic |
| Mishra [103] | Energy | x | | | | Heuristic |
| Arianyan[16] | SLA & Energy | x | | | | Heuristic |
| Gholipour[55] | SLA & Energy | x | | | | Heuristic |
| Mishra [104] | SLA & Energy | x | | | | Heuristic |
| Xu [153] | SLA & Energy | x | x | | | Heuristic |
| Azizi [18] | Energy | x | x | | x | Heuristic |
| Jangiti [75] | Energy | x | x | | x | Heuristic |
| Thiam [137] | Energy | x | x | | | Heuristic |
| Saadi [123] | Energy | x | x | | x | Heuristic |
| Gupta [58] | Energy | x | x | | | Heuristic |
| Zheng [164] | SLA | x | x | x | | MIP |
| Lin [93] | SLA | x | x | x | x | ACO |
| Guérout [57] | SLA & Energy | x | x | x | x | SS |
| Ramamoorthy [119] | SLA | x | x | | | GA |
| Sofia [125] | SLA & Energy | x | | | | GA |
| Abohamama [9] | Energy | x | | | | GA |
| Abdel-Basset [6] | Bandwidth | | | | x | WOA |
| Alresheedi [14] | SLA & Energy | x | x | x | | SSA & SCA |
| Vozmediano[108] | SLA | x | | | | SVM & QT |
| Peng [115] | SLA & Energy | x | | | | DQN |
| Barrett [22] | SLA | x | x | x | | Q |
| Ghobaei-Arani [54] | SLA | x | x | x | x | Q |
| Moghaddam [80] | SLA | x | x | | | DQN |
| Rao [120] | SLA | x | x | | x | Q |
| Bitsakos [30] | SLA | x | x | x | | DQN |
| Bibal [29] | SLA | x | x | | | SARSA |
| Jamshidi [74] | SLA & Energy | x | x | | | Q |
| Bu [35] | SLA | x | x | | | Q |
| Arabnejad [15] | SLA | x | | | | Q & SARSA |
| Ren [122] | SLA & Energy | x | x | x | | DQN |
| Ren [157] | SLA | x | x | x | | DQN |
| Nouri [112] | SLA | x | x | x | | Q |
| Witanto [149] | SLA & Energy | x | | | | Multiple |
| Sniezynski [132] | SLA & Energy | x | | | | ML |
| Padala [113] | SLA | x | | | | CT |
| Wang [146] | SLA | x | | | | CT |

that they use the CPU as the primary metric [16] and most adapt at the VM level by performing addition, removal, migration or consolidation of VMs. Approaches typically do not factor in other resources such as memory and network bandwidth. Thus, additional research into multi-objective resource allocation could be helpful.

Table 3.1 summarises some of the current MAs and classifies them based on the objectives of the approach, the considered resource and the technique used to adapt the cloud resource infrastructure. A key aspect of MAs is the technique used to perform the decision making process, and while heuristics are common in the literature, they have challenges with reacting to changes in cloud workloads. Other techniques included metaheuristics, control theory and machine learning. Generally, many of the approaches in the literature use a heuristic-based MA to determine resource provisioning and adaptation decisions. However, cloud workloads are typically heterogeneous with different QoS requirements. Heuristic MAs require pre-defined knowledge of how allocation should be done, and this can limit their use in cloud resource management [127]. Additionally, heuristic MAs do not guarantee optimal solutions for ample search space and dense networks [37]. While ML models have shown to be effective in their application in cloud resource management, a limitation of these approaches is that they require offline training and thus require retaining to take advantage of new data. In contrast, RL can operate online model-free, a valuable property in heterogeneous environments that exhibit large variability, such as cloud environments. Therefore, RL approaches can be a promising technique for MAs and would benefit from further research.

We hypothesise that the proposed hybrid MF can be paired with a MA to utilise its features and lower SLA violations further. We investigate and develop a MA that can learn dynamically how to perform the VM migration and balance achieving SLAs and energy consumption. Additionally, the proposed MA can learn when nodes can escalate a VM migration request outside their neighbouring overlay to speed the resolution of situations when a node is stressed and reduce SLA violations. We present this approach in contribution 4, Paper 4 [67], in Chapter 4.

This chapter includes a survey that extends our earlier survey in Paper 1 and provides an updated review of MAs in the literature. In particular, it focuses on recent advances in the usage of ML techniques, which are effective in managing cloud environments. ML approaches have gained prominence in cloud resource management, as they typically do not require predefined knowledge of how the allocation of resources

should be done, which enables ML to adapt to the dynamic nature of cloud environments.

### 3.3.5 Evaluation of MAs and MFs

While various approaches are being used to research MAs and MFs, evaluating these approaches on actual cloud infrastructure can be time-consuming, and cost-prohibitive [46]. Running experiments on public cloud systems requires computing, storage and network resources and security features, and it may be hard to achieve repeatability.

Simulation offers a viable solution to these problems. The performance and the characteristics of MAs and MFs can be evaluated in detail before taking on the overhead of running experiments in public cloud systems. Several simulation tools are in the literature [36, 138, 84, 92]. These vary in their features, from the programming language used, ability to support different networking models, support for SLA calculations, support for incorporating an energy consumption model, availability of a graphical user interface and accuracy of metric collection. The choice of the simulator to use is likely to be familiarity with the programming language and specific features related to the intended experiments.

# Chapter 4

# Collection of Published Papers

This chapter presents the core contributions of the thesis in the form of five published and in submission papers. The published papers appear in conference proceedings and journals with a high impact factor. The papers are summarised as follows:

- **Paper 1, provides contribution 1: Adaptation in Cloud Resource Configuration: A Survey**. A survey of resource reconfiguration in a cloud context, including a definition for cloud adaptation and classification that we use to survey the literature.
- **Paper 2, provides contribution 2: SHDF - A Scalable Hierarchical Distributed Framework for Data Centre Management**. A highly scalable cloud management framework, which can manage a large cloud data centre spanning thousands of nodes.
- **Paper 3, provides contribution 3: A Hierarchical Decentralized Architecture to enable Adaptive Scalable Virtual Machine Migration**. An empirical evaluation of the hybrid MF compared to hierarchical, decentralized and centralized MFs.
- **Paper 4, provides contribution 4: Scalable Virtual Machine Migration using Reinforcement Learning**. To address the issue with heuristic-based approaches, we propose an RL method, which can integrate well into our hybrid M and achieve fast convergence and lower SLA violations.
- **Paper 5, provides contribution 5: Dynamic Threshold Setting for VM Migration** We propose an approach capable of learning the CPU utilization to use for VM migration. We use RL to dynamically set the CPU threshold level.

## 4.1 Adaptation in Cloud Resource Configuration: A Survey

Abdul R Hummaida, Norman W Paton and Rizos Sakellariou

**Publishing state:** Published. Journal of Cloud Computing, 2016, SpringerOpen, Pages 1-16, DOI: https://doi.org/10.1186/s13677-016-0057-9

**Summary**: This paper presents a literature review of cloud adaptation to highlight features and approaches to identify open challenges and facilitate future developments. We present a definition of cloud systems adaptation, a classification of the critical features and a survey of adapting cloud resources. Based on our analysis, we identify three open research challenges: characterising the workload type, accurate online profiling of workloads, and building highly scalable adaptation mechanisms. In light of this, the remainder of the research in the thesis addresses the development of mechanisms for adapting VM placement at scale.

Chapter 3 includes an updated survey on management algorithms and management frameworks, particularly on recent advances in the usage of ML techniques in cloud resource management. We discuss the updated survey and the findings in more detail in Chapter 3.

**Key contributions:** Contribution 1 (see Section 1.4).

REVIEW

# Adaptation in Cloud Resource Configuration: A Survey

Abdul R Hummaida*, Norman W Paton
and Rizos Sakellariou

**Abstract**

With increased demand for computing resources at a lower cost by end-users, cloud infrastructure providers need to find ways to protect their revenue. To achieve this, infrastructure providers aim to increase revenue and lower operational costs. A promising approach to addressing these challenges is to modify the assignment of resources to workloads. This can be used, for example, to consolidate existing workloads; the new capability can be used to serve new requests or alternatively unused resources may be turned off to reduce power consumption. The goal of this paper is to highlight features, approaches and findings in the literature, in order to identify open challenges and facilitate future developments. We present a definition of cloud systems adaptation, a classification of the key features and a survey of adapting compute and storage configuration. Based on our analysis, we identify three open research challenges: characterising the workload type, accurate online profiling of workloads, and building highly scalable adaptation mechanisms.

**Keywords:** Autonomic cloud; cloud adaptation; resource management; elasticity

## 1 Introduction

Cloud computing is an established paradigm for providing on demand computing services to a wide range of users, including enterprises, software developers and researchers. Infrastructure Providers (IPs) manage the base infrastructure, including servers, storage and network connectivity, and typically present this infrastructure as Virtual Machines (VMs). Other providers rent these resources and become value-added resellers (VARs) as Platform as a Service (PaaS) or Software as a Service (SaaS).

VARs utilise clouds to lower operating costs by only paying for computing resources they use. The ability to expand to additional resources means they do not have to build capacity upfront. In return for these benefits, VARs typically pay a higher per hour cost for resources used, compared to managing infrastructure directly. IPs, on the other hand, have the challenge of providing these benefits to VARs. IPs build the capacity to cope with increasing demands for computing resources, which requires significant investment in infrastructure, skilled personnel and incurs power costs. Furthermore, with increased competition and commoditisation of cloud services, IPs are under pressure to reduce their prices. Amazon reduced its prices on 41 different occasions in the last few years [1]. The adoption of cloud computing does, however, open up a new market for IPs, where they can run a wide variety of computing requests that previously were housed in private infrastructure.

IPs generate revenue by meeting Service Level Agreements (SLAs). To achieve this, one approach is to periodically *Adapt* the infrastructure configuration. Adaptation typically entails a decision to increase or reduce cloud resource allocation to a workload. For example, the CPU share allocated to a VM running a web server can be reconfigured to a lower share, if SLAs can remain unaffected. The gained capacity can be used to accept new workloads or to reduce power consumption, resulting in an increase in IP profit.

This paper surveys resource reconfiguration, covering 40+ publications that focus on adaptation of computing resources in a cloud context. The chosen publications appeared in cloud focused journals and conferences. Our contributions are a definition for cloud adaptation and a classification that we use to survey the literature. To focus the scope of this work, we chose to cover adaptation of compute and storage resources. However, we recognise the potential impact adaptation of network resource can play. For example, multiple under-utilised network routers can be powered down

*Correspondence: abdul.hummaida@postgrad.manchester.ac.uk
University of Manchester, School of Computer Science, M13 9PL, UK
Manchester, UK
Full list of author information is available at the end of the article

by reconfiguring the network infrastructure, thus lowering power consumption.

Several surveys pull together results of different features of cloud resource management. In [2] the authors surveyed elastic approaches in cloud computing, providing a high level overview of the approaches. Our survey is different as it investigates adaptation and, as we demonstrate later, adaptation is a superset of elasticity. In [3], the authors comprehensively discuss approaches to efficient data centres, choosing to focus on power consumption. Our work covers power as an adaptation objective and also covers SLA and revenue. In [4], the authors surveyed autoscaling, and classified the literature based on the adaptation techniques used. Their work focused on the Infrastructure as a Service (IaaS) client's perspective, while we focus on the IaaS provider, thus their work excluded VM migration and server consolidation. In [5], the authors provide an overview of the mechanisms and techniques employed to manage elasticity from the perspective of a SaaS provider, while we focus on the IaaS provider. In [6], the authors investigate cloud resource management and in [7], the authors present common aspects used in cloud computing environments, such as metrics, tools and strategies. In [8], the authors surveyed the VM allocation problem and models and algorithmic approaches. In [9], the authors present analysis of autonomic resource management in general, and specifically Quality of Service aware autonomic resource management. In [10], the authors surveyed SLA-based cloud research including the techniques used for adaptive resource allocation. In [11], the authors surveyed cloud computing elasticity using a classic systematic review covering metrics and tools. In [12], the authors summarised different method and theory used in cloud resource allocation and monitoring. In [13], the authors depict a broad literature analysis of resource management in the cloud. While there is some overlap from these surveys with our work, they chose a different classification scheme to our work, which focuses on adaptation of resource configuration, enabling us to analyse the factors that influence the adaptation process. Additionally we investigate factors affecting scalability of the various proposals in the literature. To the best of our knowledge there is no other work that uses our chosen dimensions.

As PaaS can be built on top of IaaS, there can be similarities between how resources are adapted in both environments. However, as IaaS is typically presented at the VM abstraction level, IPs have less visibility into the nature of workloads and their configurations. This presents additional challenges for Autonomic [14] approaches to adapting resources on IaaS. Section 2 introduces cloud infrastructure and lays the foundation for a discussion on how this can be adapted in Section 3; we also define the dimensions used in the survey. Section 4 surveys the literature based on the adapted cloud resource, identifying the techniques and approaches used. Section 5 presents open challenges in adapting resource configuration and Section 6 presents our conclusions.

## 2 Cloud Systems setup

Cloud computing is defined as *"a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*[15]. In this section we introduce the constituents of cloud systems.

*Compute Resource:* The core processing capabilities that are used to execute software instructions. We define this as comprising of a CPU, typically in multicore configuration, CPU cache and primary storage memory. Data centres typically house many thousands of servers containing these compute resources.

*Storage Resource:* Non-volatile secondary storage memory houses the data used by compute resources. As this resource is typically cheaper than primary memory, many operating systems are able to use it as an extension of main memory, to temporarily swap out unused memory state. Many data centres will have servers with access to internal storage as well as to a Storage Area Network that consolidate and abstracts the complexity of accessing storage throughout the data centre.

*Network Resource:* includes the network cards that connect into servers as well as infrastructure components that include repeaters, load balancers, switches and firewalls. Networks can use different topologies and protocols, which influence the level of security, resilience and Quality of Service.

*Virtual Resource:* is an abstraction added onto compute, storage and network resources. It enables slicing of these resources into smaller chunks that can be scaled vertically or horizontally. Typically virtualisation is used in a data centre to slice data centre compute resource into Virtual machines, and potentially to present several logical processors by mapping these onto a single physical processor. Network cards

and storage are also virtualised and presented as individual devices to VMs.

*Service Management Resource (SMR)*: is a knowledge library where IPs store management objectives, policies, pricing and orchestration information.

*Management Tools*: are used by IPs to provision, monitor, reconfigure, back up and restore the infrastructure.

IPs typically build the infrastructure and offer access to virtual resources, with a VM being the main component. VMs reside on physical nodes of heterogeneous capabilities where the performance characteristics of compute, storage and network vary. Demand for recourses varies over time as users consume and release these resources. As more resources are used, power consumption in the data centre increases and IPs may choose to optimise the allocation of VMs to physical nodes. In the next section, we will cover IPs objectives and approaches used to optimise this allocation.

## 3 Cloud Systems Adaptation

In this section we introduce the IPs objectives and approaches to adapting the cloud infrastructure.

To meet workload demands, IPs can use Elasticity [16] to reconfigure resources in an autonomic manner. The limitation of this view is that it assumes the IP's objective is to satisfy precisely all workload demands. While this may be true, it may not always be the case, as the IP has finite resources and may apply differentiation on requests. Additionally, the IP may decide it is more cost effective to pay a penalty for an SLA violation instead of scheduling the request. The current view on Elasticity abstracts several complex activities. We refine this view by separating the decision making process from how the cloud environment is reconfigured, by defining elasticity as the *on demand ability, to scale vertically or horizontally segmented resources in discrete units*. To achieve a specific business goal, IPs go through a decision making process that changes the infrastructure, a process we name Cloud Systems Adaptation. We define this as a *change to provider revenue, data centre power consumption, capacity or end-user experience where decision making resulted in a reconfiguration of compute, network or storage resources*. Reconfiguration is the process of increasing or reducing resource allocation to a workload, through elasticity.

Core to cloud systems adaptation is a decision making process that decides the resources to reconfigure and how. Figure 1 shows the inputs into the decision making process, including:

1 The desired management objective in each adaptation cycle from the SMR.
2 The adaptation techniques and infrastructure metrics.

When decision making is complete, Elasticity is used to scale the infrastructure resources.

We define the dimensions of cloud systems adaptation as: 1) *Adapted cloud resource*, which categorises what resources are modified and how; 2) *Adaptation objective* is a desired business outcome; 3) *Adaptation techniques* are a set of analytical and modelling techniques used to achieve the adaptation objective; 4) *Adaptation engagement* categorises when the adaptation process is invoked; 5) *Decision engine architecture* categorises the different architectures used by the decision making engines within the literature; 6) *Managed infrastructure type* categorises whether node capabilities and properties are used in the decision making. These dimensions are presented in Table 1 and discussed in the following subsections.

### 3.1 Adapted Resource

We extend the definition of possible resource adaptation from [17] in Table 1, which describes our classification of the literature and the dimensions used. *VM* level adaptation are typically applied to improve/reduce workload performance due to an increased/reduced demand by adjusting *CPU, memory, disk bandwidth* and/or *storage*. For example, a web server running on a VM may need a bigger share of CPU due to an increased number of requests.

*Node* level adaptation could be applied to add capacity by *powering on* a node. Power consumption could be reduced by using Dynamic Voltage and Frequency Scaling (*DVFS*) [18], before the node is *powered off* when not needed. Node configuration can also be adapted when a VM's requirements extend beyond the capacity of its hosting node, so that it needs to be *migrated* to another node that has the required capacity. Migration can also be used to reduce power consumption, by consolidating VMs into fewer nodes and enabling some nodes to be switched off.

*Cluster* level adaptation is applied to facilitate node adaptation and to adhere to any reliability policies used by IPs by *adding* and/or *removing* nodes.

### 3.2 Adaptation objective

All of the proposals surveyed drive adaptation to minimise *SLA* violations and some trade this off with a secondary objective. Examples include reducing *power*

consumption, maximising IP *revenue* and combined where multiple objectives are sought. A small number of proposals focus on reducing the *customer cost* of using the infrastructure.

### 3.3 Adaptation technique

Several adaptation techniques have been applied to cloud infrastructure in the literature, including *Heuristic*, *Control theory* or *Machine learning* [19].

Heuristic based adaptation techniques use problem specific knowledge to provide a quick solution and trade preciseness of the outcome with lower time complexity, which makes them good candidates for dynamic resource allocation on the cloud. Control theory can provide QoS guarantees by using a feedback controller, that dynamically adjusts the behaviour of the system based on the measured outputs. Machine Learning techniques are grouped into two categories, supervised and unsupervised learning.

### 3.4 Adaptation Engagement

Cloud systems adaptation needs to be invoked in order to evaluate the infrastructure and determine whether resource reconfiguration is required. The approaches used in the literature fall onto *Reactive*, *Proactive* and *Hybrid* engagement.

Reactive approaches invoke adaptation when a monitored metric, e.g. CPU utilisation, reaches a specific threshold.

Proactive approaches predict what demands will be placed on the infrastructure and invoke adaptation ahead of the predicted resource contention point.

Hybrid approaches utilise proactive approaches and combine these with reactive approaches, as way to engage adaptation for long and short term time scales.

### 3.5 Decision Engine Architecture

The architecture of the decision engine governs where the engine is placed and how it operates. *Centralised* architectures use an engine with a global view of the managed infrastructure and can adapt resource across the entire infrastructure.

*Hierarchical* architectures typically divide the infrastructure into multiple clusters, placing an engine (Level 1) in each cluster. A global, Level 2, engine coordinates each of the Level 1 engines.

*Distributed* architectures typically use a Peer-to-Peer protocol [20] that enables nodes to communicate directly without a centralised controller.

### 3.6 Managed Infrastructure

Cloud systems are typically diverse and made of a *heterogeneous* set of compute and storage resources [21]. Some of the proposals incorporate the type of the managed infrastructure in the decision making process, while other proposals assume a *homogeneous* infrastructure, where every node has the same capability and power consumption.

## 4 Adaptation in Cloud Resource Configuration

In this section we survey the literature that adapt cloud resource configuration, focusing on compute and storage resources. We chose to focus on the reconfigured resource and classify the reviewed literature on this dimension, in order to analyse patterns that are specific to a cloud resource. The reconfigured resources are:

1 CPU and Memory
2 VM Migration
3 Node Power Usage
4 Storage

In general, proposals apply cloud systems adaptation to minimise SLA violations and some trade this off with a secondary objective, by recognising that meeting SLAs is not the only business objective for IPs. To achieve this, proposals use different techniques and engage the adaptation at different points. Additionally, the execution complexity of the proposals impact their ability to scale their approach on data centres with thousands of nodes. Therefore, the *secondary objectives*, *adaptation techniques*, *adaptation engagement* and *decision engine architecture* distinguish the various proposals in their adaptation of cloud resource configuration. The remainder of this section will be structured principally according to the adapted resource, and then within each resource following the remaining dimensions in Table 1. Table 2 provides a summary comparison of the literature, in an IaaS context.

### 4.1 **VM Adaptation - CPU and Memory**

As core computing resources, CPU and memory adaptation have been widely researched. Many of the proposals scale the infrastructure horizontally by adding new VMs, typically via predefined VM classes [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]. While this is simpler to apply, compared to fine grain CPU and memory configuration, it may lead to wastage by over-allocating resources to workloads as well consume more power. In [33] the authors further argued that fine grain CPU and memory configuration reduces the provisioning overhead and mitigates SLA violations. Other proposals, particularly those focusing on maximising revenue, apply fine grain management of VM resources with CPU and memory configurations modified in discrete values using the Xen [34] hypervisor API. In [35, 36, 37], the authors utilise Xen's credit-based CPU scheduler to set the CPU share for workloads and in [38, 37], the authors additionally utilise

Xen's ability to define the amount of memory assigned to each VM. The life cycle management of workloads can be categorised into two overlapping phases. Admission control [62], which is the decision to accept a new workload if it contributes to the current management objectives and resource adaptation [10], which reconfigures the infrastructure after a state change. Several proposals treat admission control as distinct phase and assume availability of free resources. While this simplifies the approach, it may unnecessarily power on a new node. Alternatively admission control should be used as an opportunity to apply cloud system adaptation and redistribute existing workloads.

*4.1.1 Secondary Objectives*

Some of the proposals focus on reducing power consumption in the data centre [39, 40, 41, 42, 43]. While this has a direct impact on an IPs profits, some of the proposals aim to maximise revenue by increasing capacity to service workloads [19, 36, 44, 45]. In contrast, the authors in [23] aim to reduce the cost of using cloud infrastructure to customers on Amazon EC2 [46], by automatically allocating resources based on the current demand. The authors in [24, 47] aim to reduce the complexity in resource provisioning of the Apache Hadoop framework [48], by enabling automated allocation of resources and configuration parameters, and minimise the incurred infrastructure cost. Both approaches attempt to predict the workload behaviour to optimise run time performance, however they differ in their methodology. The authors in [24] used offline training, while the authors in [47] used historical data from past jobs. The latter approach may initially produce lower optimal allocations as it builds job performance history. However, over time this could enable the approach to build better clusters of workload signatures that enable it to make more optimal allocations. Therefore there is a tradeoff between initial performance and time taken to build workload knowledge. While offline approaches can be used to improve the online decision making process by constructing a model of the system behaviour, this has an upfront overhead and is not practical to apply for every application deployed on IaaS.

To reduce power consumption of a node before turning it off, proposals [3, 40] use power management features in modern nodes (DVFS) to scale down both the frequency of the CPU and the voltage used. An alternative approach to DVFS was used in [42], where the authors incorporate the power cost and priority of a VM in the decision of where to add the VM, thus reducing the number of active nodes. Figure 2 shows the components that may get adapted on compute resources.

*4.1.2 Adaptation Technique*

A common adaptation technique is Control theory [23, 25, 26, 35, 41, 49, 44, 50], which aims to guarantee system stability by adapting resource configurations at defined intervals. Some of the control theory proposals react to monitored metrics such as CPU utilisation thresholds and workload throughput [50], but most of the proposals surveyed use a proactive mechanism to forecast the future workloads, typically using time series. In contrast, [51] proposed a control theory based approach that utilised Fuzzy Logic to predict short term CPU utilisation. The authors in [52] also used a reactive Fuzzy controller, which can handle conflicting rules. The authors approach attempts to simplify the complexity of setting thresholds by using imprecise thresholds such as *high* and *low* for specifying elasticity rules. However this requires human experts to set multiple values for the approximate thresholds.

Following a heuristic approach, the authors in [53] assign application tiers to nodes, preserving CPU utilisation in each node below a 60% threshold. A local search optimises the initial allocation, guided by availability guarantees. This architecture results in the heuristic being invoked at three different time-scales, evaluating different adaptation decisions on each time period. In [45], the authors propose a lightweight heuristic based on workload response time, which is defined by the customer. Their approach attempts to satisfy workload response time by incrementally adding CPU and memory resources to the workload. The authors approach requires a deployment portal, but does uniquely account for customer specified constraints, such as budget, when making adaptation decisions. In later work, the authors [54] use an open queueing network, a queueing theory technique, to reduce utilisation cost to customers. Their approach identifies and scales a bottlenecked tier in a multi-tier cloud application. The authors in [32] simplify the cloud application to a typical request queueing model and combine this with binary search.

The heuristic based approach in [55] allows control and adaptation of multiple resources simultaneously, by building groups of resources and performance metrics, which can be adapted based on customer defined events. While the approach uses multiple objective optimisation, the authors did not show empirical evidence of their approach or its ability to scale.

Machine Learning based proposals fall into two categories, supervised learning [24, 56] and unsupervised learning [38]. Given the variability of workloads deployed on cloud infrastructure, Reinforcement Learning (RL) seems promising as it does not rely on preconstructed models of the controlled infrastructure, by discovering system behaviour online without prior

training. The main disadvantage of RL is the online training time, which can be exponential to the size of the explored space, potentially resulting in poor decisions during the learning phase. To tackle this challenge, [38] proposed combining RL with a Simplex method to reduce the search space to a smaller valuable set, and then used online CPU and memory utilisation to guide decision making.

Utility based approaches are used to define a measure of usefulness towards a management objective, typically utilising a customer metric like response time as objective to the utility function. Proposals typically build utility frameworks by constructing a performance model of multi-tier applications embedded in an optimisation problem [19, 22, 25, 36, 42], where a utility function expresses satisfaction of each workload towards assigned resources. Utility has also been combined with control theory in [41, 44] to apply a fine grain configuration of CPU and memory, by estimating the benefits of potential adaptations and incorporating a notion of risk. In contrast, the authors in [49] argue that defining multiple levels of QoS, Q-states, beyond the traditional minimum level, is easier for customers to define than utility functions. The challenge with utility based approaches is humans could find it difficult to define the utility functions needed in a complex system.

*4.1.3 Adaptation Engagement*

Proposals adapt CPU and memory configuration either by reacting to a breached metric or by forecasting a metric change. Reactive approaches typically set and monitor a utilisation threshold to CPU and memory [38]. However, setting the optimal threshold is not simple and typically requires workload knowledge. Some proposals used experimentation [39, 57] to set threshold values. In [45], the authors proposed an alternative approach, where the customer defines the SLA and the IPs set the resource utilisation thresholds. The proposals in [19, 50] react to workload response time, instead of CPU/Memory thresholds, and trigger adaptation to preserve response times to the requested levels. In addition to the challenge of setting the threshold level, reactive approaches risk oscillating system state by reacting too frequently to varying node utilisation. In [55], the authors tackle this by using four thresholds and two duration periods to track for how long the threshold has been reached.

Proactive mechanisms are typically time series based, where a sequence of events at defined intervals are analysed to find patterns that can be used to forecast future values. Time series estimators include Auto Regressive Moving Average (ARMA) [58], Smoothing Spline [59], Kalman Filter [60] and Fast Fourier Transform [61]. Proposals that forecast workload arrival rate have an additional challenge to map this to a utilisation forecast. Several proposals tackle this by using an offline phase [24, 37, 44, 49] to build performance models of workloads, which are then used to make adaptation decisions based on online CPU and memory measurements. A disadvantage of the offline approach is it may have a significant overhead and may not cope with dynamic behaviour of some workloads.

A few proposals combine both proactive and reactive approaches in a hybrid approach to engage the adaptation process. In [29], the authors propose a proactive controller that provisions based on peak load seen in the last hour, with a reactive controller for sudden bursts, but it had no ability to scale down CPU/memory resources. In [26, 62], the authors extend this approach and use a reactive controller for scaling up and a proactive controller for scaling down, by removing whole VMs. The authors claim this hybrid approach is able to cope with sudden bursts as well as being able to conserve energy by proactively switching nodes off. The authors did not experiment with gradual scaling of CPU frequency and voltage using techniques such as DVFS, which typically reduces power consumption. The authors in [52] combine both proactive time series analysis and a reactive fuzzy controller. The authors approach attempts to simplify the complexity of setting thresholds by using imprecise thresholds such as *high* and *low* for engaging adaptation. However this requires human experts to set multiple values for the approximate thresholds.

*4.1.4 Decision Engine Architecture*

The scalability of a proposal is primarily affected by the execution complexity of the adaptation decision making process. Most proposals are centralised, and memory and CPU adaptation are scheduled across the entire infrastructure. While this gives opportunities for global optimisation, it presents a significant challenge when managing thousands of resources. In [38], the authors used a centralised reinforcement learning engine and the time taken to stabilise performance increased with the size of the managed cluster. The central controller in [41] took significant time to execute the scheduling of 15 nodes, which had $10^9$ control options, just to adapt CPU resource - memory configuration was not covered. The centralised engine in [19] was only able to manage 400 nodes with 1000 VMs, when adapting CPU and VM configurations. In later work [53], the authors changed their centralised approach to a hierarchical architecture, resulting in the ability to support 7,200 servers with up to 60,000 VMs. In [35], the authors propose an alternative layered approach where each node has a decision engine, with no global controller. While this enabled each node to

perform its own allocation, it lost out on the opportunity to redistribute workloads across the data centre infrastructure.

To improve on scalability of the centralised approaches, researchers investigated decentralised approaches such as hierarchical and distributed frameworks. In [44], the authors proposed hierarchical controllers and divided the infrastructure into multiple clusters, where each cluster is managed by a local controller. The hierarchical controllers run at different intervals, with a local cluster controller running more frequently than a global controller. In [36], the authors chose to slice the hierarchy along the operations of the controllers. A Level 1 controller handles VM placement and load balancing, and runs every 30 minutes. A Level 2 controller handles the resources of a node, and runs every few minutes. The challenges with hierarchical approaches include choosing the run time interval of the global controller and the lack of an escalation path between the local and global controllers. Therefore in a sudden burst scenario, a workload may exhibit SLA violations before the global level controller is engaged. An additional challenge is limiting the size of each cluster so it does not become too large for the controller to manage, thus encountering the same challenge as centralised approaches.

Distributed approaches typically focus on VM consolidation and will be covered as part of our analysis of *VM migration*, in the next subsection.

### 4.2 **Node Adaptation - VM Migration**

Nodes maybe adapted when a VM's requirements extends beyond the capacity of its hosting node, and it needs to be moved to another node that has the required capacity. Proposals opting to simplify their approach assume the entire infrastructure is homogeneous and has the same computing capability and power consumption, which may lead to suboptimal VM migration decisions. Proposals that do take the infrastructure capability into account, usually focus on power consumption of nodes. Some proposals assume the ability to capture the relationships between cooperating VMs, and many proposals abstract how workload KPIs, like response time, can be captured. Such proposals are better suited to PaaS, where a deeper integration between the workload and infrastructure is available, and workload metrics and configurations can be made available to the decision-making engine.

#### *4.2.1 Secondary Objectives*

Proposals apply VM migration primarily to minimise SLA violations, and some proposals aim to reduce power consumption as a secondary objective, by consolidating workloads and switching nodes off, as evidenced by the summary in Table 2.

VM migration adds an overhead and can impact the SLA of the migrated VM and other VMs on the cooperating nodes, yet this is considered acceptable [63] given the opportunities migration can present. In [39], the authors argue that CPU power consumption is the largest contributor to a node's power consumption, thus VM migration can be used to lower power consumption.

#### *4.2.2 Adaptation Technique*

Beloglazov et al [39] used a heuristic based adaptation technique and explored three policies, minimisation of migrations (MM), highest potential growth and Random choice, and concluded the MM policy can achieve significant energy savings, compared to non-energy aware policies. The authors argue there is a minor SLA violation trade off, to achieve these energy savings. The MM policy selects VMs with the highest CPU utilisation to migrate to another node. A disadvantage of this approach is it migrates VMs that are already at risk of SLA violation, due to the CPU utilisation, and further increases the risk by adding the cost of live migration. In [43, 64], the authors use a heuristic implemented as a peer-to-peer protocol, enabling nodes to communicate directly without a centralised controller. Two cooperating nodes determine whether to migrate a VM based on the defined objectives. While [43] did not take into account the cost or duration of the conflict before applying the migration, [64] incorporated migration cost into the decision making. In contrast to other proposals, the authors in [44] incorporate the power consumption of the decision engine. Other proposals include VMware's Distributed Resource Scheduler (DRS) [65], which uses greedy hill-climbing to reduce cluster imbalance. DRS incorporates migration cost and benefit, based on workload demands observed in the last hour. Similarly, a greedy heuristic that incorporates migration cost was proposed in [27]. The authors in [31] aimed to reduce the number of nodes used migration as well as reduce VM migration times at the same time, by using a multi-objective Genetic Algorithm based on hybrid group encoding.

In contrast to heuristic based proposals, the authors in [22, 66] uniquely formulated VM migration as a Constrained Satisfaction Problem, taking into account the migration overhead. Tchana et al [66] combine VM migration with Software migration, by collocating several software applications on the same VM to reduce the number of VMs used. The authors claim significant reduction in power consumption can be achieved by

using this approach. However, a limitation of this approach is it requires explicit knowledge of the software being migrated, compared to VM migration, which typically abstracts the software within a VM.

Similar to [44], the authors in [22] used utility as measure the satisfaction of each managed workload and a global decision module prioritises decisions that maximise a global utility.

A less common adaptation technique for VM migration is time series analysis, proposed in [40], to predict contention for resources through a Fast Fourier Transform algorithm. The authors engaged the migration before it is needed, and minimise cost by only migrating when the resource contention is predicted to last beyond a defined period of time. In a multi-adaptation technique, Zhu et al [67] experimented with integrating a fuzzy logic controller with a trace-based controller, arguing the integration resulted in better resource allocation compared to the non-integrated approach.

Proposals typically do not cover cloud system adaptation during admission control phase, assuming availability, however the authors in [68] migrate VMs during the admission control phase, by using a heuristic solution based on hill climbing search techniques.

*4.2.3 Adaptation Engagement*

To engage VM migration, the authors in [39] used a two-threshold reactive approach. The low threshold aims to lower power consumption and triggers VMs to be migrated off a node, which is then set to sleep mode. The high threshold aims to meet SLA and triggers migration of a VM with the highest utilisation to another node. The double threshold approach takes a snapshot in time of the current CPU utilisation and thus can suffer from false positives caused by workload utilisation peaks and troughs. In later work, Beloglazov et al [69] proposed an adaptive auto-adjustment of the upper threshold, based on statistical analysis of historical data collected during the lifetime of VMs, tackling statistical outliers in their earlier approach. Similarly, the authors in [70] proposed a dynamic threshold approach that finds and adjusts thresholds at runtime. Zuo et al [71] also use an adaptive threshold. The authors monitor 3 metrics: number of resource requests, resource service capacity and resource service strength, and propose a dynamic weighted evaluation, dividing the resource load into three states including Overload, Normal and Idle.

Proactive approaches [40, 44] start the VM migration before the conflict occurs, to avoid sustained service degradation from the cost of the migration. In [44], the authors proposed performing a cost and benefit analysis before applying migration, and only invoked a migration if the benefit outweighed the cost of the migration.

*4.2.4 Decision Engine Architecture*

Most proposals are centralised and VM migration is scheduled across the entire managed infrastructure. While this gives opportunities for global optimisation, it presents a significant challenge when managing thousands of resources. Despite its name, VMware's Distributed Resource Scheduler [65] uses a centralised load balancing approach to engaging VM migration, so it suffers the same scalability challenges of centralised approaches proposed in academia. Zheng et al [31] aim to reduce the number of nodes used in migration as well as reduce VM migration times, by using a multi-objective Genetic Algorithm based on hybrid group encoding. The approach used a centralised controller and limited simulation to only 200 nodes. Additionally, the authors did not explore the time complexity of their Genetic algorithm.

To improve the scalability of a centralised approach, researchers investigated hierarchical and distributed frameworks.

Hierarchical approaches tackle the scalability challenge by reducing the frequency of engaging the global controller. The hierarchical approach in [22] used a local decision module for each application and a global decision module. Application satisfaction is regularly measured using a utility function and communicated to the global module, which prioritises requests to satisfy a global utility. An alternative approach was proposed in [67], where an additional Level 3 (L3) controller was used to manage multiple clusters operating at seconds (L1), minutes (L2) and days (L3) intervals. However the authors did not explore the scalability of their approach.

For a distributed and decentralised approach to managing the data centre, the authors in [43, 64] proposed a peer-to-peer protocol that enables nodes to communicate directly without a centralised controller. A periodic node discovery service enables nodes to find new neighbouring nodes to communicate with. On each round of the protocol, two cooperating nodes determine to migrate a VM based on defined objectives. The distributed approaches in [43, 64] are used to redistribute the load across the cluster as well consolidate VMs. Using simulation, the authors claim their approaches can manage more than 100,000 nodes. A challenge with distributed approaches is the lack of a global view of the infrastructure, which impact the ability to reach a globally optimal solution. Additionally, gossip approaches consume considerable bandwidth to implement propagation of node state across the entire data centre infrastructure .

### 4.3 **Node Adaptation - Power**

Proposals adapt a node's power configuration to reduce operational costs for IPs. Proposals may use a policy in the VM placement phase to use the most energy-efficient nodes first, apply power management features on a node and eventually migrate VMs and switch the node to a sleep state. To reduce the power consumption of a node before turning it off, some proposals use Dynamic Voltage and Frequency Scaling (DVFS), which is a framework to change the frequency and/or operating voltage of nodes based on system performance requirements. To utilises DVFS, it needs to be supported by both the node and OS. Modern processors typically support multiple levels of frequency/voltage, which can be selected through the OS. Proposals typically select a frequency/voltage level that reduces the node capability and minimises impact to workloads, applying a trade-off between workload performance and power consumption.

An alternative approach to DVFS was used in [42], where the authors incorporate the power cost and priority of a VM in the decision of where to add the VM, thus reducing the number of active nodes. While DVFS has been widely deployed and proven to reduce power consumption, the authors in [72] argue that DVFS can have an impact on mult-tier application performance. They propose a solution to minimise the impact, by increasing the DVFS adjustment frequency and predicting the workload burst cycle.

#### *4.3.1 Adaptation Techniques*

Proposals differ in their approach to reducing node power consumption, with some researchers opting to migrate VMs and set the node to a sleep state [39, 44, 56, 67], compared to incrementally reduce power consumption by using DVFS.

Beloglazov et al [39] propose a heuristic to consolidate workloads and switch nodes into a sleep state, arguing that an idle node can consume 70% of the power consumed by a node running at the full CPU speed. Their approach was able to switch a node to sleep mode within 20 seconds. However, the authors did not discuss how nodes can be be woken up from sleep mode if more nodes are required to service requests. Other proposals that do not utilise DVFS include the gossip based protocol in [43], which places new VM requests on the highest loaded node capable of hosting it. VMs are redistributed by moving a VM from a lower loaded to a higher loaded node if it can be hosted. In [64], the authors take into account power consumption in the decision making. The authors in [31] aim to reduce the number of nodes used in migration as well as reduce VM migration times at the same time, by using a multi-objective Genetic Algorithm based on hybrid group encoding.

In contrast, [19, 40, 73] utilise DFVS to gradually reduce power consumption and switch nodes to a sleep state. The authors in [40] use time series, while [19, 73] use a heuristic to adjust DVFS.

#### *4.3.2 Adaptation Engagement*

To adapt power configuration, reactive approaches [69, 56] use a low threshold for CPU utilisation to switch nodes to sleep state. In contrast, proactive approaches predict workload utilisation and switch nodes to sleep state at the predicted time intervals. In [41], the authors used a Kalman filter to predict the number of requests. VM capability and power consumption were captured offline, by measuring the average response times achieved when different CPU shares were assigned to the VM. The authors modelled risk in the decision making to cater for the cost of switching nodes on and off, arguing this reduces SLA violations considerably compared to a non risk aware controller. Core to this argument is SLA violations, or opportunity cost, in having to power on a node. However, with commoditisation of Solid state storage (SSD), which offers significant boot performance compared to Hard disk drives, many servers use SSD to boot the operating system. The authors previous conclusions may need to be revisited to re-evaluate whether more nodes using SSD can be left in switched off mode and switched on nearer to the time they are needed. Similarly, [40, 53] proactively adjust the node frequency and eventually switch the node to sleep state.

#### *4.3.3 Decision Engine Architecture*

To consolidate VMs, proposals migrate VMs between nodes by searching for suitable nodes that can take additional VMs without violating another management objective. As the scalability of migrating VMs was covered in the Node Adaptation subsection, here we focus on the approaches to managing power reduction at large scale.

In the gossip based protocol in [43], the authors experimentally assessed the power consumption of the proposal, by measuring the number of active servers. However they do not incorporate an explicit notion of power cost in their policy. In [64], when two nodes communicate they attempt to consolidate all VMs onto one peer and the released peer is set into the power saving mode. If the VMs cannot be entirely consolidated onto one node, the protocol attempts to redistribute the load across the two nodes, taking into account power consumption and migration cost.

Proposals utilising DVFS to lower power consumption typically use a centralised decision engine [40, 73, 19], although Addis et al proposed a hierarchical architecture in later work [53].

### 4.4 Storage Adaptation

Cloud storage adaptation can be applied to both I/O access and the storage itself, although this area is less covered compared to other cloud resources.

*4.4.1 Adaptation Technique*

Control theory is used by researchers to adapt different levels of the storage stack. Padala et al [35] used an application controller to determine disk I/O resources needed at the node level. While the approach can apply service differentiation, it over-allocates disk I/O bandwidth when these are available, which potentially increases power consumption. In [74], the authors used control theory to adapt the central storage tier, focusing on the Hadoop Distributed File System, from a customer perspective. Offline profiling data was used to build the transfer function into the constructed system model, combining this with online CPU metrics from the storage node.

Another technique used to adapt I/O access is supervised machine learning, proposed in [24], focusing on automated provisioning of Hadoop jobs.

*4.4.2 Adaptation Engagement*

To engage storage adaptation, the approach in [74] reacts to the CPU utilisation of the storage node. The first controller adds and removes storage nodes and a second controller rebalances data across the new set of storage nodes. To ascertain some of the thresholds, the authors used offline experimentation with Cloudstone benchmark. In contrast, the proactive approach in [24] used a two phase approach, where phase one is offline and builds a prediction model using past job information and a k-medoid clustering and support vector machine. Phase two is online and uses a staging area to obtain a resource utilisation signature for newly submitted jobs. These signatures are then matched to the offline constructed data for the decision making process. The authors assume availability of job history information, and the staging area imposes additional costs that have to be met by either the IPs or end users. In contrast, the authors in [35] used a second order ARMA model, taking into account two previous control intervals to predict workload performance, by using response time as the performance metric.

*4.4.3 Decision Engine Architecture*

The scalability of centralised approaches is typically problematic [43], however proposals in [35, 24] do not migrate VMs to resolve contention, therefore do not require a global view of the infrastructure. This places less emphasis on the scalability of their approaches.

The centralised proposal in [74] needs to rebalance data when nodes join and leave a storage cluster. During the rebalancing phase, no additional adaptation can be carried out. The impact of this limitation will increase as the number of nodes in the cluster increase, thus limiting the applicability of the approach.

## 5 Open Research Challenges

While there has been considerable research in adaptation of resource configuration, there are several open challenges. Based on our analysis, the following are open challenges in cloud systems adaptation, in an IaaS context:

1 Many of the proposals in the literature focus on managing web/multi tired applications, as can be seen on Table 2, and use application metrics as input into the decision making process. Other proposals attempt to manage generic workload types and typically utilise threshold based approaches to trigger adaptation. A potentially better approach is to characterise the *workload type* and engage adaptation that takes into account the workload type. Several projects attempt to analyse and characterise cloud workloads. Analysis of public Google traces [21, 75, 76, 77] has shown variance in the resources utilised and the duration of cloud tasks, making popular simplifications such as being able to slot workloads on resources unsuitable [21]. Additionally, users typically overestimate resources reservations, leading to significant wastage [76]. Some existing approaches aim to predict future workloads using classical prediction models such as ARMA [28], a linear regression model [78] and a hybrid model tuned to bursty web traffic [32, 79]. Other characterisation approaches aim to predict workload resource utilisation, by identifying a feature of the workload. The authors in [80] match applications with appropriate VM types by defining application profiles, which are manually extracted from workflow logs. The authors in [77] classify tasks based on resource utilisation and the authors in [81, 82] extract utilisation usage signatures. The authors in [31] use a load predictor that clusters historical resource utilisation, and select the cluster set with the highest similarity as a training sample into a Neural Network. However, these approaches simplify the impact of colocating VMs, which can lead to significant performance overhead [83, 84]. The authors in [85] tackle colocation interference and perform four parallel classifications on each application to evaluate the impact of vertical and horizontal scale, server configuration, and the impact of colocating applications. However this approach needs specific knowledge of the application in order to profile and classify.

Based on the current state of art, there is no generic non application aware online classification of workload types, which are typically deployed on IaaS. A generic mechanism to predict whether the workload is a user desktop, web server, file server or batch job, can enable the decision engine to adapt resource configuration in an optimal way for the workload type. This can potentially allow the workload to complete quicker or conserve resource otherwise not utilised by the workload, and enable colocation of VMs in a way that does not introduce interference.

2 *Offline profiling* and staging area approaches are typically used to experimentally derive workload resource requirements. However this has an upfront overhead and is not practical to apply for every application that will be deployed on a IaaS. Several proposals have attempted online profiling and/or monitoring of workloads, however these typically require explicit knowledge of the application [85], or an output from the VM such as latency or response time [82, 86, 87], which is typically not available to IPs. More research is need into application agnostic mechanisms that can extract workload resource requirements, and impact of adaptation, dynamically at run time.

3 *Scalability* of computing systems is an understood challenge in traditional enterprise infrastructure. However cloud environments magnify this challenge due to the larger size and heterogeneity of infrastructure used in cloud data centres. Table 2 shows a summary of the proposals in the literature, including the number of nodes each proposal attempted to manage. This shows many of the proposals do not explore the scalability of their approach and typically implement a centralised decision engine. Some of the proposals explore scalability of managing several thousand nodes, which is still significantly below many modern data centres, which can house more than 100,000 nodes [88]. Proposals that explored scalability capable of managing modern data centres tend to implement a distributed decision engine. However these approaches trade off ability to manage a large infrastructure with a reduction in optimal resource allocation. Additionally, these approaches consume considerable bandwidth for the nodes to communicate directly across the entire infrastructure. More research is required to demonstrate robust and practical application of distributed approaches, which can achieve similar level of optimal allocation as centralised approaches.

## 6 Conclusion

This paper presented a definition of cloud systems adaptation and a classification of the key features. We analysed the literature and highlighted approaches and techniques used to enable adaptation of cloud resource configuration.

Workload management on IaaS entails controlling the admission of new workloads and periodically adapting resource configuration to achieve a management objective. Proposals in the literature aim to minimise SLA violations and some trade this off with a secondary objective, such as reducing power consumption or maximising IP revenue. To achieve these objectives, several adaptation techniques have been used. The architecture of the decision engine has a significant impact on the scalability of a proposal, with centralised approaches not being able to scale on large data centres. While there has been considerable research, we have highlighted several open challenges that are worthy of further investigation.

**Competing interests**
The authors declare that they have no competing interests.

**Author's contributions**
AH carried out the survey of the literature, drafted the manuscript and identified open research challenges. NP and RS provided insight and guidance in developing the structure and dimensions for the literature classification, critically reviewed the paper and suggested additional papers to investigate. All authors read and approve the final manuscript.

**Author's Information**
Abdul Hummaida is a PhD candidate in the School of Computer Science, University of Manchester, UK. He completed his bachelor's degree on Software Engineering from the School of Computer Science, University of Manchester, UK. He is currently working on scalability of cloud management systems. His research interests include cloud computing, autonomic computing and workload management. He is also a Director of Software Engineering at Appsense.
Norman Paton is a Professor of Computer Science at the University of Manchester, where he co-leads the Information Management Group. He works principally on databases and distributed information management. Current research interests include pay-as-you-go data integration, sensor query processing and infrastructures for adaptive systems development. He also works on genome data management, in particular exploring the use of data integration techniques for making better use of experimental and derived data in systems biology. He has been an investigator on over 40 research grants from the UK research councils, the EU and industry, and has published around 200 refereed articles.
Rizos Sakellariou is with the School of Computer Science at the University of Manchester, UK, where he carries out research in the broad area of parallel and distributed systems while at the same time he enjoys teaching and never stops to be amazed by university politics. He has published over 100 research papers in the area.

**References**

1. Jassy, A.: Amazon Web Services Summit. https://aws.amazon.com/summits/san-francisco/ Accessed May 2016
2. Galante, G., Bona, L.C.E.d.: A survey on cloud computing elasticity. In: Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing. UCC '12, pp. 263–270. IEEE Computer Society, Washington, DC, USA (2012)
3. Beloglazov, A., Buyya, R., Lee, Y.C., Zomaya, A.: A taxonomy and survey of energy-efficient data centers and cloud computing systems. Advances in Computers **82**, 47–111 (2011)

4. Botran, T.L., Miguel-Alonso, J., Lozano, J.A.: Auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing **12**(4), 559–592 (2014)
5. Najjar, A., Serpaggi, X., Gravier, C., Boissier, O.: Survey of Elasticity Management Solutions in Cloud Computing. Computer Communications and Networks, pp. 235–263. Springer, 236 Gray's Inn Road, Floor 6, London WC1X 8HB, UK (2014)
6. Jennings, B., Stadler, R.: Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management **23**(3), 567–619 (2015)
7. Coutinho, E.F., Carvalho Sousa, F.R., Rego, P.A.L., Gomes, D.G., Souza, J.N.: Elasticity in cloud computing: a survey. annals of telecommunications - annales des télécommunications **70**(7), 289–309 (2014). doi:10.1007/s12243-014-0450-7
8. Mann, Z.A.: Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. ACM Comput. Surv. **48**(1), 11–11134 (2015). doi:10.1145/2797211
9. Singh, S., Chana, I.: Qos-aware autonomic resource management in cloud computing: A systematic review. ACM Comput. Surv. **48**(3), 42–14246 (2015). doi:10.1145/2843889
10. Faniyi, F., Bahsoon, R.: A systematic review of service level management in the cloud. ACM Comput. Surv. **48**(3), 43–14327 (2015). doi:10.1145/2843890
11. Naskos, A., Gounaris, A., Sioutas, S.: Cloud Elasticity: A Survey. In: Karydis, I., Sioutas, S., Triantafillou, P., Tsoumakos, D. (eds.) Algorithmic Aspects of Cloud Computing: First International Workshop, ALGOCLOUD 2015, Patras, Greece, September 14-15, 2015. Revised Selected Papers, pp. 151–167. Springer, Cham (2016)
12. Mohamaddiah, M.H., Abdullah, A., Subramaniam, S., Hussin, M.: A survey on resource allocation and monitoring in cloud computing. International Journal of Machine Learning and Computing **4**(1), 31–38 (2014)
13. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing: Issues and challenges. Journal of Grid Computing **14**(2), 1–48 (2016)
14. Murch, R.: Autonomic Computing. IBM Press, 1 New Orchard Rd, Armonk, NY 10504, US (2004)
15. NIST: Sp 800-145: Definition of cloud computing. Technical report, NIST, 100 Bureau Drive, Gaithersburg, USA (Sep 2011). NIST. http://csrc.nist.gov/publications/PubsSPs.html Accessed May 2016
16. Herbst, N.R., Kounev, S., Reussner, R.: Elasticity in cloud computing: What it is, and what it is not. In: 10th International Conference on Autonomic Computing, pp. 23–27 (2013)
17. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for cloud infrastructure management. Future Generation Computer Systems **29**(2), 472–487 (2013)
18. Magklis, G., Semeraro, G., Albonesi, D.H., Dropsho, S.G., Dwarkadas, S., Scott, M.L.: Dynamic frequency and voltage scaling for a multiple-clock-domain microprocessor. IEEE Micro **23**, 62–68 (2003)
19. Addis, B., Ardagna, D., Panicucci, B., Zhang, L.: Autonomic management of cloud service centers with availability guarantees. In: 2010 IEEE 3rd International Conference on Cloud Computing, pp. 220–227. IEEE, Washington, DC, USA (2010)
20. Sedaghat, M., Hernández-Rodriguez, F., Elmroth, E.: Autonomic resource allocation for cloud data centers: A peer to peer approach. In: IEEE International Conference on Cloud and Autonomic Computing, pp. 131–140. IEEE, Washington, DC, USA (2014)
21. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing. SoCC '12, pp. 7–1713. ACM, New York, NY, USA (2012). doi:10.1145/2391229.2391236. http://doi.acm.org/10.1145/2391229.2391236
22. Van, H.N., Tran, F.D., Menaud, J.-M.: Sla-aware virtual resource management for cloud infrastructures. In: IEEE International Conference on Computer and Information Technology, vol. 02, pp. 357–362. IEEE, Washington, DC, USA (2009)
23. Bodík, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., Patterson, D.: Statistical machine learning makes automatic control practical for internet datacenters. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing. HotCloud'09. USENIX Association, Berkeley, CA, USA (2009)
24. Lama, P., Zhou, X.: Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In: Proceedings of the 9th International Conference on Autonomic Computing. ICAC '12, pp. 63–72. ACM, New York, NY, USA (2012)
25. Malkowski, S.J., Hedwig, M., Li, J., Pu, C., Neumann, D.: Automated control for elastic n-tier workloads based on empirical modeling. In: Proceedings of the 8th ACM International Conference on Autonomic Computing. ICAC '11, pp. 131–140. ACM, New York, NY, USA (2011)
26. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: 2012 IEEE Network Operations and Management Symposium, pp. 204–212. IEEE, Washington, DC, USA (2012)
27. Zhani, M.F., Cheriton, D.R., Zhang, Q., Simon, G., Boutaba, R.: Vdc planner: Dynamic migration-aware virtual data center embedding for clouds. In: IEEE International Symposium on Integrated Network Management, pp. 18–25. IEEE, Washington, DC, USA (2013)
28. Roy, N., Dubey, A., Gokhale, A.: Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In: IEEE International Conference on Cloud Computing, pp. 500–507 (2011)
29. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier internet applications. ACM Transactions on Autonomous and Adaptive Systems **3**(1) (2008)
30. Celaya, J., Sakellariou, R.: An adaptive policy to minimize energy and sla violations of parallel jobs on the cloud. In: IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 507–508. IEEE, Washington, DC, USA (2014)
31. Zheng, S., Zhu, G., Zhang, J., Feng, W.: Towards an adaptive human-centric computing resource management framework based on resource prediction and multi-objective genetic algorithm. Multimedia Tools and Applications, 1–18 (2015)
32. Zhang, Q., Chen, H., Shen, Y., Ma, S., Lu, H.: Optimization of virtual resource management for cloud applications to cope with traffic burst. Future Generation Computer Systems **58**, 42–55 (2016). doi:10.1016/j.future.2015.12.011
33. Dawoud, W., Takouna, I., Meinel, C.: Elastic virtual machine for fine-grained cloud resource provisioning. Global Trends in Computing and Communication Systems **269**, 11–25 (2011)
34. Citrix: Xen. http://www.xenserver.org Accessed May 2016
35. Padala, P., Hou, K.-Y., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A.: Automated control of multiple virtualized resources. In: Proceedings of the 4th ACM European Conference on Computer Systems. EuroSys '09, pp. 13–26. ACM, New York, NY, USA (2009)
36. Almeida, J., Almeida, V., Ardagna, D., Cunha, Í., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing **70**, 344–362 (2010)
37. Fargo, F., Tunc, C., Al-Nashif, Y., Akoglu, A., Hariri, S.: Autonomic workload and resource management of cloud computing services. In: IEEE International Conference on Cloud and Autonomic Computing, pp. 101–110. IEEE, Washington, DC, USA (2014)
38. Bu, X., Rao, J., Xu, C.-Z.: Model-free learning approach for coordinated configuration of virtual machines and appliances. In: 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 12–21. IEEE, Washington, DC, USA (2011)
39. Beloglazov, A., Abawajyb, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems **28**, 755–768 (2012)
40. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing. SOCC '11, pp. 5–1514. ACM, New York, NY, USA (2011)
41. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., Jiang, G.: Power and performance management of virtualized computing environments via lookahead control. In: Autonomic Computing ICAC, pp. 3–23. IEEE, Washington, DC, USA (2008)
42. Cardosa, M., Korupolu, M.R., Singh, A.: Shares and utilities based

power consolidation in virtualized server environments. In: 11th IFIP/IEEE International Conference on Symposium on Integrated Network Management, pp. 327–334 (2009)
43. Wuhib, F., Stadler, R., Spreitzer, M.: Dynamic resource allocation with management objectives: implementation for an openstack cloud. IEEE Transactions on Network and Service Management **9**(2), 213–225 (2012)
44. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: International Conference on Distributed Computing Systems, pp. 62–73. IEEE, Washington, DC, USA (2010)
45. Han, R., Guo, L., Ghanem, M.M., Guo, Y.: Lightweight resource scaling for cloud applications. In: International Symposium on Cluster, Cloud and Grid Computing, pp. 644–651. IEEE, Washington, DC, USA (2012)
46. Amazon: AWS. http://aws.amazon.com/ec2/ Accessed May 2016
47. Koehler, M.: An adaptive framework for utility-based optimization of scientific applications in the cloud. Journal of Cloud Computing: Advances, Systems and Applications **3**, 4 (2014)
48. Apache: Hadoop. http://hadoop.apache.org Accessed May 2016
49. Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-clouds: Managing performance interference effects for qos-aware clouds. In: Proceedings of the 5th European Conference on Computer Systems. EuroSys '10, pp. 237–250. ACM, New York, NY, USA (2010)
50. Zhu, X., Wang, Z., Singhal, S.: Utility-Driven Workload Management Using Nested Control Design. In: American Control Conference. IEEE, Washington, DC, USA (2006)
51. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. Cluster Computing **11**, 213–227 (2008)
52. Jamshidi, P., Ahmad, A., Pahl, C.: Autonomic resource provisioning for cloud-based software. In: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS 2014, pp. 95–104. ACM, New York, NY, USA (2014)
53. Addis, B., Ardagna, D., Panicucci, B., Squillante, M.S., Zhang, L.: A hierarchical approach for the resource management of very large cloud platforms. IEEE Transactions on Dependable and Secure Computing **10**, 253–272 (2013)
54. Han, R., Ghanem, M.M., Guo, L., Guo, Y., Osmond, M.: Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. Future Generation Computer Systems **32**, 82–98 (2014)
55. Hasan, M.Z., Magana, E., Clemm, A., Tucker, L., Gudreddi, S.L.D.: Integrated and autonomic cloud resource scaling. In: Network Operations and Management Symposium, pp. 1327–1334. IEEE, Washington, DC, USA (2012)
56. Berral, J.L., Goiri, I.n., Nou, R., Julià, F., Guitart, J., Gavaldà, R., Torres, J.: Towards energy-aware scheduling in data centers using machine learning. In: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. e-Energy '10, pp. 215–224. ACM, New York, NY, USA (2010)
57. Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Resource pool management: Reactive versus proactive or lets be friends. Computer Networks: The International Journal of Computer and Telecommunications Networking **53**, 2905–2922 (2009)
58. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control, 4th edn. John Wiley & Sons Inc, 111 River Street Hoboken, NJ 07030-5774 (2008)
59. Boor, C.D.: A Practical Guide to Splines, 1st edn. Springer, 233 Spring Street, New York, NY 10013-1578, USA (2001)
60. Kalman, R.E.: A new approach to linear filtering and prediction problems. Journal Fluids Eng. **82**, 35–45 (1960)
61. Loan, C.V.: Computational Frameworks for the Fast Fourier Transform. Society for Industrial and Applied Mathematics, 3600 Market Street, 6th Floor, Philadelphia, PA (1987)
62. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation Computer Systems **26**, 871–879 (2011)
63. Voorsluys, W., Broberg, J., Venugopal, S., Buyya, R.: Cost of virtual machine live migration in clouds: A performance evaluation. In: Proceedings of the 1st International Conference on Cloud Computing. CloudCom '09, pp. 254–265. Springer, Berlin, Heidelberg (2009)
64. Sedaghat, M., Hernández-Rodriguez, F., Elmroth, E., Girdzijauskas, S.: Divide the task, multiply the outcome: Cooperative vm consolidation. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 300–305. IEEE, Washington, DC, USA (2014)
65. Gulati, A., Shanmuganathan, G., Holler, A., Ahmad, I.: Cloud-scale resource management: Challenges and techniques. In: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing. HotCloud'11, pp. 3–3. USENIX Association, Berkeley, CA, USA (2011)
66. Tchana, A., Palma, N.D., Safieddine, I., Hagimont, D., Diot, B., Vuillerme, N.: Euro-par 2015: Parallel processing: 21st international conference on parallel and distributed computing, vienna, austria, august 24-28, 2015, proceedings, 305–316 (2015)
67. Zhu, X., Young, D., Watson, B.J., Wang, Z., Rolia, J., Singhal, S., McKee, B., Hyser, C., Gmach, D., Gardner, R., Christian, T., Cherkasova, L.: 1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center. In: International Conference on Autonomic Computing, pp. 172–181. IEEE, Washington, DC, USA (2008)
68. Casalicchio, E., Menascé, D.A., Aldhalaan, A.: Autonomic resource provisioning in cloud systems with availability goals. In: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference. CAC '13, pp. 1–1110. ACM, New York, NY, USA (2013)
69. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience **24**, 1397–1420 (2012)
70. Choi, H.W., Kwak, H., Sohn, A., Chung, K.: Autonomous learning for efficient resource utilization of dynamic vm migration. In: Proceedings of the 22Nd Annual International Conference on Supercomputing. ICS '08, pp. 185–194. ACM, New York, NY, USA (2008)
71. Zuo, L., Shu, L., Dong, S., Zhu, C., Zhou, Z.: Dynamically weighted load evaluation method based on self-adaptive threshold in cloud computing. Mobile Networks and Applications, 1–15 (2016). doi:10.1007/s11036-016-0679-7
72. Wang, Q., Kanemasa, Y., Li, J., Lai, C.A., Matsubara, M., Pu, C.: Impact of dvfs on n-tier application performance. In: Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems. TRIOS '13, pp. 5–1516. ACM, New York, NY, USA (2013)
73. Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P., Zhu, X.: Delivering energy proportionality with non energy-proportional systems: Optimizing the ensemble. In: Proceedings of the 2008 Conference on Power Aware Computing and Systems. HotPower'08, pp. 2–2. USENIX Association, Berkeley, CA, USA (2008)
74. Lim, H.C., Babu, S., Chase, J.S.: Automated control for elastic storage. In: Proceedings of the 7th International Conference on Autonomic Computing. ICAC '10, pp. 1–10. ACM, New York, NY, USA (2010)
75. Di, S., Kondo, D., Cappello, F.: Characterizing cloud applications on a google data center. In: Parallel Processing (ICPP), 2013 42nd International Conference On, pp. 468–473. IEEE, Washington, DC, USA (2013)
76. Moreno, I.S., Garraghan, P., Townend, P., Xu, J.: An approach for characterizing workloads in google cloud to derive realistic resource utilization models. In: Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium On, pp. 49–60. IEEE, Washington, DC, USA (2013)
77. Zhang, Q., Zhani, M.F., Boutaba, R., Hellerstein, J.L.: Dynamic heterogeneity-aware resource provisioning in the cloud. IEEE Transactions on Cloud Computing **2**(1), 14–28 (2014). doi:10.1109/TCC.2014.2306427
78. Yang, J., Liu, C., Shang, Y., Cheng, B., Mao, Z., Liu, C., Niu, L., Chen, J.: A cost-aware auto-scaling approach using the workload prediction in service clouds. Information Systems Frontiers **16**(1), 7–18 (2013). doi:10.1007/s10796-013-9459-0
79. Liu, C., Shang, Y., Duan, L., Chen, S., Liu, C., Chen, J.: Optimizing Workload Category for Adaptive Workload Prediction in Service Clouds. In: Barros, A., Grigori, D., Narendra, C.N., Dam, K.H. (eds.) Service-Oriented Computing: 13th International Conference, ICSOC

2015, Goa, India, November 16-19, 2015, Proceedings, pp. 87–104. Springer, Berlin, Heidelberg (2015)
80. Chard, R., Chard, K., Bubendorfer, K., Lacinski, L., Madduri, R., Foster, I.: Cost-aware elastic cloud provisioning for scientific workloads. In: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference On, pp. 971–974. IEEE, Washington, DC, USA (2015)
81. Gong, Z., Gu, X., Wilkes, J.: Press: Predictive elastic resource scaling for cloud systems. In: Network and Service Management (CNSM), 2010 International Conference On, pp. 9–16. IEEE, Washington, DC, USA (2010)
82. Zhang, L., Zhang, Y., Jamshidi, P., Xu, L., Pahl, C.: Service workload patterns for qos-driven cloud resource management. Journal of Cloud Computing **4**(1), 1–21 (2015). doi:10.1186/s13677-015-0048-2
83. Xu, F., Liu, F., Jin, H., Vasilakos, A.V.: Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. Proceedings of the IEEE **102**(1), 11–31 (2014). doi:10.1109/JPROC.2013.2287711
84. Feller, E., Ramakrishnan, L., Morin, C.: Performance and energy efficiency of big data applications in cloud environments: A hadoop case study. Journal of Parallel and Distributed Computing **79–80**, 80–89 (2015). doi:10.1016/j.jpdc.2015.01.001. Special Issue on Scalable Systems for Big Data Management and Analytics
85. Delimitrou, C., Kozyrakis, C.: Quasar: Resource-efficient and qos-aware cluster management. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS '14, pp. 127–144. ACM, New York, NY, USA (2014). doi:10.1145/2541940.2541941. http://doi.acm.org/10.1145/2541940.2541941
86. Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S., Koziris, N.: Automated, elastic resource provisioning for nosql clusters using tiramola. In: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium On, pp. 34–41. IEEE, Washington, DC, USA (2013)
87. Naskos, A., Stachtiari, E., Gounaris, A., Katsaros, P., Tsoumakos, D., Konstantinou, I., Sioutas, S.: Dependable horizontal scaling based on probabilistic model checking. In: Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium On, pp. 31–40. IEEE, Washington, DC, USA (2015)
88. Miller, R.: Data Center Knowledge. http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/ Accessed May 2016

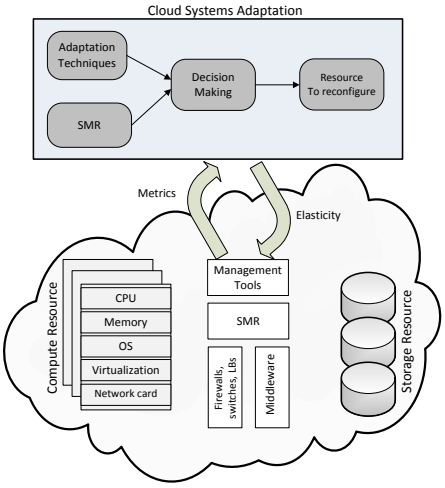



**Figure 1** Cloud Systems Adaptation sequence

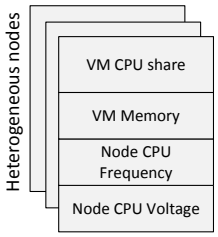



**Figure 2** Compute Resource components

**Table 1** Dimensions for Cloud Systems Adaptations

| Dimension | Definition |
|---|---|
| Adapted Resource | VM $\subset$ {Adjust CPU, Memory, Storage, Disk Bandwidth}<br>Node $\subset$ {Power on/off, Adjust DVFS, Migrate VM}<br>Cluster $\subset$ {Add/remove nodes} |
| Adaptation Objective | SLA, Power, Revenue, Customer Cost |
| Adaptation Technique | Heuristics, Control theory, Queing theory Machine learning |
| Adaptation Engagement | Reactive, Proactive, Reactive/Proactive |
| Decision Engine Architecture | Central, Hierarchical, Distributed |
| Managed Infrastructure | Heterogeneous, Homogeneous |

**Table 2** Summary of literature that adapt cloud resources, ordered by the Decision Engine Architecture.

Legend: CT= Control Theory; RL= Reinforcement learning; CSP= Constrained satisfaction problem; ML= Supervised machine learning; P2P= Peer-to-Peer QT= Queueing Theory; GA= Genetic Algorithm; TS= Time series; R= Reactive; P= Proactive; Hom=Homogenous; Het= Heterogeneous

| **Project** | **Objective** | | | | | **Resource** | | | | | | | **Tech** | **Adapt trigger** | **Arch** | **Infra** | **Workload** | **Setup [#nodes]** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | P | SLA | Rev | Cust cost | *Whole VM/node* | *CPU* | *Mem* | *Migrate* | Disk I/O | DVFS | Node off | ST | | | | | | |
| Zheng[31] | x | x | | | x | | | x | | | | | GA | P | Central | Hom | Generic | Simulation[200] |
| Zhang [32] | | x | | | x | | | | | | | | QT | P | Central | Hom | Multi tier | Simulation |
| Zuo[71] | x | x | | | x | | | x | | | x | | Heuristic | R | Central | Het | Generic | Simulation |
| Tchana [66] | x | | x | | x | | | x | | | | | CSP | R | Central | Het | Generic | Private + AWS |
| Beloglazov[39, 69] | x | x | | | x | | | x | | | x | | Heuristic | R | Central | Het | Generic | Simulation [100] [800] |
| Wesam[33] | | x | | | | x | x | | | | | | Heuristic | R | Central | Het | Multi tier | Xen test bed |
| Gmach[57] | | x | | | | | | x | | | x | | CT | R | Central | Hom | Generic | Simulation |
| Fargo[37] | x | x | | | | x | x | | | x | x | | Heuristic | P | Central | Hom | Web App | Xen test bed |
| Won Choi[70] | | x | | | | | | x | | | | | Heuristic | R | Central | Hom | Generic | Linux test bed |
| Iqbal[62] | | x | | | | | | x | | | x | | Heuristic | R + P | Central | Hom | Generic | Eucalyptus |
| Roy[28] | | x | x | | x | | | | | | | | CT | P | Central | Hom | Multi tier | NA |
| Xiangping Bu[38] | | x | | | | x | x | | | | | | RL | R | Central | Hom | Multi tier | Xen test bed |
| Padala[35] | | x | | | | x | | | x | | | | CT | P | Layered | Hom | Multi tier | Xen test bed |
| Xu[51] | | x | | | | x | | | | | | | CT | P | Central | Hom | Web App | ESX test bed |
| Jamshidi[52] | | x | | x | x | | | | | | | | CT | R + P | Central | Hom | Web App | Azure |
| Bodik[23] | | x | | x | x | | | | | | | | CT | P | Central | Hom | Multi tier | Simulation |
| Lama[24] | | | | x | x | | | | | | | | SML + Heuristic | P | Central | Het | Hadoop | ESX test bed |
| Koehler[47] | | | | x | x | | | | | | | x | Utility | P | Central | Hom | Hadoop | KVM test bed |
| Kusic[41] | x | x | | | x | x | | | | | x | | CT+ Utility+TS | P | Central | Het | Multi tier | ESX test bed |
| Zhu[50] | | x | | | | x | | | | | x | | CT + Utility | R | Central | Hom | Web App | HP-UX |
| Hasan[55] | | x | | | | x | | | | | | | Heuristic | R | Central | Hom | Generic | Test bed |
| Cardosa[42] | x | | | | x | | | | | | | | Utility + Heuristic | R | Central | Hom | Generic | ESX test bed |
| Shen[40] | x | x | | | | x | x | x | | x | | | TS | P | Central | Het | Web App | Xen test bed |
| Nathuji[49] | | x | | | | x | | | | | | | CT | P | Central | Het | Generic | Hyper-V test bed |
| Malkowski[25] | | x | | | x | | | | | | | | CT + Heuristic | P | Central | Hom | Multi tier | Xen test bed |
| Lim[74] | | x | | | | | | | | | | x | CT | R | Central | Hom | Hadoop | Xen test bed |
| Ali-Eldin[26] | | x | | | x | | | | | | | | CT | R + P | Central | Hom | Generic | Simulation |
| Zhani[27] | x | | x | | x | | | x | | | | | Heuristic | R | Central | Hom | Generic | Simulation [400] |
| Han[45] | | x | x | | x | x | x | | x | | | | Heuristic | R | Central | Hom | Generic | IC Cloud |
| Han[54] | | x | | x | x | | | | | | | | QT | R | Central | Hom | Generic | Simulation |
| Gulati[65] | | x | | | | x | x | x | x | | | | Greedy Heuristic | R | Central | Het | Generic | ESX test bed |
| Berral[56] | x | x | | | | | | x | | | x | | SML | P | Central | Hom | Generic | Simulation [400] |
| Addis[19] | | x | x | | | x | | x | | x | x | | Utility + Heuristic | R | Central | Het | Multi tier | IBM test bed |
| Urgaonkar [29] | | x | | | x | | | | | | | | QT | R + P | Central | Hom | Multi tier | Xen test bed |
| Tolia [73] | x | x | | | | | | x | | x | | | Heuristic | R | Central | Hom | Generic | Xen test bed |
| Casalicchio [68] | | x | x | | | | | x | | | | | Heuristic | N/A | Central | Hom | Generic | Workstation |
| Celaya [30] | x | x | | | x | | | | | | x | | Heuristic | P | Central | Hom | Parellel | Simulation |
| Addis[53] | x | x | x | | | x | | x | | x | x | | Utility + Heuristic | P | Hierarch | Het | Multi tier | IBM test bed [7200] |
| Zhu[67] | | x | | | | x | | x | | | x | | CT + Heuristic + TS | P | Hierarch | Hom | Web App | ESX/Simulation |
| Jung[44] | x | x | | | | x | | x | | | x | | Heuristic + Utility+TS | P | Central + Hierarch | Het | Multi tier | Xen test bed |
| Almeida[36] | | x | x | | | x | | | | | x | | Utility | P | Hierarch | Hom | Multi tier | Simulation |
| Nguyen Van[22] | | x | x | | x | | | x | | | | | Utility + CSP | R | Hierarch | Het | Generic | Simulation |
| Sedaghat[64] | x | | | | | | | x | | | | | Heuristic+ P2P | R | Distrib | Het | Generic | Simulation [100,000] |
| Wuhib[43] | x | x | | | x | | | x | | | | | Heuristic + P2P + TS | P | Disrib | Hom | Generic | Simulation [160,000] |

## 4.2 SHDF - A Scalable Hierarchical Distributed Framework for Data Centre Management

Abdul R Hummaida, Norman W Paton and Rizos Sakellariou

**Publishing state:** Published. In Proceedings of 16th International Symposium on Parallel and Distributed Computing (ISPDC), 2017.
DOI: https://doi.org/10.1109/ISPDC.2017.15

**Summary**: A fundamental hypothesis in this PhD project is that the benefits of hierarchical and decentralized architectures can be combined, and their weaknesses mitigated. This paper presents a novel hybrid hierarchical decentralized management framework that rapidly provides the information needed for scaling decision making, to address the scalability challenges in centralized controllers, which have been shown to have scalability challenges. The centralized controller in [87] would need to search and evaluate a significant number of configurations, $10^9$, to manage the small infrastructures used in the experiments. The proposed MF migrates VMs by starting with neighbouring nodes, followed by nodes in the same cluster and then nodes in other clusters. Therefore the escalation process seeks proximity during migration. This is particularly useful in preserving the performance of multi VM applications and reducing the migration distance. The proposed MF provides the mechanisms for migration and additional constraints can be provided by the chosen MA, used in conjunction with the proposed MF. An example constraint could be to only migrate VMs within the cluster in a multi VM application deployment. To achieve this, the MA can perform a cost benefit analysis of migrating outside the cluster versus a delay in performing the migration.

We evaluate the performance of the approach by simulation and demonstrate that it is a viable solution for managing large data centres through rapid information dissemination and the ability to make decisions using a global view. In our evaluation, the SLAs apply equally to all VMs.

To examine the impact of migration instability, a VM being migrated several times during its lifetime, we evaluated the stability of the proposed MF (Section 4.3). This shows the hybrid MF performs fewer migrations of the same VM compared to other architectures. This is achieved through autonomous decision making in each node. While the Hybrid MF can reduce VM migration instability, it does not have explicit

instability detection and mitigation. This can be achieved by extending the decision making in the MA to include cost benefit analysis and prediction of the future state of migration targets.

The computational cost of an MF is typically comprised of the execution of metric collection and execution of the decision making. The metric collection in the proposed hybrid MF is done autonomously by each node in the infrastructure. It takes O(log N) rounds to reach all nodes, where N is the number of selected cooperating nodes in an overlay. Evaluation of the computational cost of decision making shows the hybrid MF spends milliseconds in central components, compared to 100s seconds in the hierarchical MF, resulting in an inherently scalable proposal.

Chapter 3 discussed the fault-tolerance of the different architectures, and how decentralized MFs have fault tolerance built into their design. Within the proposed approach, a Lead Node (LN) acts as an ordinary execution node as well as a management node within the infrastructure. This dual role enables fewer dedicated management nodes, which reduces the number of nodes in the proposed MF compared to approaches in the literature.

Using the gossip protocol, nodes regularly exchange their view of the capacity of all the nodes within their overlay. Each LN consolidates the known capacity within its cluster and shares this with other LNs it cooperates with. These metrics are shared in an aggregated form that describes the sizes of VMs that can be additionally provisioned with a given cluster. The metric sharing mechanism assumes the availability of a time synchronization mechanism, such as the Network Time Protocol, for the use of timestamps. For the evaluation, the simulated infrastructure is configured into clusters, with each cluster consisting of 1000 nodes and 20 overlays.

DCSim is used to evaluate the proposed MF and can account for node resource contention. DCSim tracks the difference between the requirements of each VM and the assigned CPU resources. VMs with lower allocation than their requirements experience a delay in their processing, which is representative of the difference in required and assigned CPU resources.

**Key contributions:** Contribution 2 (see Section 1.4).

# SHDF - A Scalable Hierarchical Distributed Framework for Data Centre Management

Abdul Rahman Hummaida
School of Computer Science
University of Manchester, UK
abdul.hummaida@postgrad.manchester.ac.uk

Norman W Paton
School of Computer Science
University of Manchester, UK
norman.paton@manchester.ac.uk

Rizos Sakellariou
School of Computer Science
University of Manchester, UK
rizos@manchester.ac.uk

***Abstract*—A promising approach to increase the efficiency of infrastructure usage is to adapt the assignment of resources to workloads. This can be used, for example, to consolidate existing workloads so that the new capability can be used to serve new requests, or alternatively unused resources may be turned off to reduce energy consumption. Many architectural solutions have been presented for data centre management, however these tend to be centralised and may suffer in their ability to scale and support data centres with tens of thousands of nodes. Distributed approaches solve the scalability problem, however these do not have a global view of resources across the data centre. To address this, we propose a novel hybrid distributed hierarchical framework that is effective at providing the information needed for decision making at scale. We evaluate the performance of our approach by simulation, and demonstrate that a hybrid approach is a viable solution for managing large data centres, through rapid information dissemination and ability to make decisions using a global view.**

## I. Introduction

Infrastructure Providers (IP) typically abstract data centre resources and present them to customers through a virtualisation layer, with a Virtual Machine (VM) as the most common form. A key challenge for IPs is to construct resource utilisation mechanisms that can scale to large data centres, which can house more than 100,000 nodes [1]. VM resource management for Infrastructure as a Service (IaaS) entails controlling the admission of new VMs, onto physical nodes and periodically adapting resource configuration to achieve a management objective. Proposals in the literature aim to minimise Service Level Agreement (SLA) violations, and some trade this off with another objective, such as reducing energy consumption or maximising IP revenue [2]. To achieve these objectives, several approaches are used, which typically rely on a central management controller for deciding how to change the configuration of the infrastructure. However, the architecture of the management controller has a significant impact on the scalability of a proposal, with centralised approaches not being able to scale to large data centres [2], as CPU and memory adaptation are scheduled across the entire infrastructure. While this gives opportunities for global optimisation, it presents a significant challenge when managing thousands of resources. The central controller in [3] took significant time to execute the scheduling of 15 nodes, which had $10^9$ control options, just to adapt CPU resource; memory configuration was not covered. The centralised engine in [4] was only able to manage 400 nodes with 1000 VMs, when adapting CPU and VM configurations.

In order to solve the scalability challenge of large data centres, hierarchical and distributed approaches have emerged as an alternative and received considerable attention recently [5], [6], [7], [8], [9], [10].

Hierarchical architectures tackle the scalability challenges of centralised approaches, by dividing the infrastructure into several levels and lowering the number of nodes handled by the management controller, thus reducing the time complexity of solving a global optimisation problem. The challenge with hierarchical architectures is dividing the infrastructure into small enough groups to enable efficient local resource allocation while remaining coordinated enough to come close to a globally optimal solution. Distributed architectures are decentralised and typically do not use a central decision making component. They enable individual nodes to cooperate directly [7], [8], [9], [10], and thus allow the approach to scale to very large data centres. One of the challenges with distributed architectures is they typically consume a high amount of bandwidth for the communication between the nodes across the whole data centre. In addition, due to reduced visibility of the whole infrastructure, distributed approaches tend to make locally plausible but globally suboptimal decisions.

We propose a Scalable Hierarchical Distributed Framework (SHDF), which overcomes the weaknesses of both approaches, by achieving high scalability and ability to apply efficient allocation. SHDF consists of hierarchical controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a *Node Controller* (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. Each NC operates in a distributed architecture and cooperates with a *Lead Node* (LN), which is a higher level controller for all the NCs within a given cluster. Our contributions are:

1) An architectural framework for data centre infrastructure, which can manage a large cloud data centre spanning thousands of nodes.
2) A new hybrid architecture, which has the scalability characteristics of distributed systems and the ability to construct a global view.
3) An empirical evaluation of the hybrid architecture, in comparison with hierarchical and distributed approaches.

The rest of this paper is organised as follows. Section 2 describes background and related work. Section 3 describes the architecture of our proposed scalable framework and Section 4 describes an implementation of the architecture. Section 5 presents an evaluation of our implementation and compares it to two other approaches: distributed and hierarchical. In Section 6 we discuss future work and draw conclusions.

## II. Background & Related Work

*1) Hierarchical Controllers:* Hierarchical approaches have been widely studied in cloud resource management. The approaches typically use a multi level hierarchical approach running at different time intervals. In Addis et al. [5] the lowest level controller runs every hour and performs VM placement, power management and workload profiling. The authors claim this can scale to 7,200 nodes with up to 60,000 VMs. In [11], the lowest level controllers manage a small number of machines and the applications that are hosted on them. At the next higher level, a controller manages machines owned by multiple lower level controllers. The authors in [12] used 3 levels, where the highest level controller managed multiple clusters operating at seconds (L1), minutes (L2) and days (L3) intervals, however the authors did not explore the scalability of their approach. The authors in [13] chose to slice the hierarchy along the operations of the controllers. A Level 1 controller handles VM placement and load balancing, and runs every 30 minutes. A Level 2 controller handles the resources of a node, and runs every few minutes. The authors in [14] focus on VM placement in their approach using a 2 level hierarchical controllers.

Hierarchical proposals typically utilise a controller running in a centralised manner within the scope of a cluster of nodes. As the number of nodes within the cluster increases, the decision making algorithm faces the same challenges as traditional centralised approaches, with the execution time resulting in an inability to react to SLA violations. Our proposed framework tackles this by reducing the operations performed by the centralised controllers and distributing management functionality to lower execution nodes, therefore providing a hybrid hierarchical distributed framework. In our approach, each node controller autonomously manages the execution node, including detection of stress states and violation of SLAs. These node controllers can cooperate with other nodes and higher level controllers to perform management of the infrastructure and workloads.

In [15], the authors use a hierarchical approach with VM migration escalation. The approach performs initial assignment of VMs to clusters (containers) and periodically, lower controllers decide what to optimise and pass the decision to parent controllers. The authors consider the time complexity of the decision making algorithm and place an upper bound of optimisations to be performed by the hierarchical controllers.

In [16], the authors outline how a collection of hierarchical autonomic managers can collaborate using messages. While the authors acknowledged the importance of scalability, they did not explore the scalability of their approach.

Our approach has similarities to [17], which proposed a framework for managing a hierarchical cloud management system. Due to the computational requirements of management nodes, the authors propose every node is either used as a management node, or an execution node, never playing both roles. This contrasts with our approach, execution nodes can also perform management roles, resulting in an fewer nodes dedicated to the management of the infrastructure. Our approach additionally enables the leaf nodes to operate in a distributed manner. Proposals typically do not build cooperation between controllers, resulting in controllers operating independently. In contrast, the authors in [12], [18], [19] proposed controller cooperation. In [12], [19], the lower level controllers propagated workload satisfaction with assigned resources to a higher controllers, which is then used in to possibly further optimise resource assignment. The authors in [18] proposed cooperating controllers, where lower level controllers can escalate a management request to a higher controller, rather than waiting for an action in the next management cycle. However the authors approach only escalated from a mid level controller within the hierarchy and did not escalate from the lowest level controllers, which our approach does.

*2) Distributed Controllers:* In [20], [8], the authors use a heuristic implemented as a peer-to-peer protocol, enabling nodes to communicate directly without a centralised controller. Two cooperating nodes determine whether to migrate a VM based on the defined objectives. While [20] did not take into account the cost or duration of the conflict before applying the migration, [8] incorporated migration cost into the decision making. A periodic node discovery service enables nodes to find new neighbouring nodes to communicate with. On each round of the protocol, two cooperating nodes determine to migrate a VM based on defined objectives. The distributed approaches are used to redistribute the load across the cluster as well consolidate VMs. Using simulation, the authors claim their approaches can manage more than 100,000 nodes. A challenge with distributed approaches is the lack of a global view of the infrastructure, which impacts the ability to reach a globally optimal solution. In contrast, our approach has the scalability characteristic of distributed systems yet has the ability to have a global view of the infrastructure through controllers at each level of the hierarchy.

The authors in [10], proposed a distributed probabilistic algorithm, using an overlay network and a decentralised load balancing technique. The authors considered scalability and impact of node availability churn, although they did not discuss the impact on energy consumption. The authors proposal is a distributed algorithm and can be integrated with our proposed SHDF.

The authors in [21] proposed a distributed approach for managing the workload of large, enterprise cloud data, focusing on reducing energy consumption and SLA violations. The authors used a *hypercube* structured overlay, with similar cost to our approach, with N nodes reaching status propagation in O(log N) rounds. The authors in [27], proposed a distributed

scheme where nodes broadcast resource requests to all nodes, during a migration scenario. In contrast, our approach has a bigger view of the infrastructure, through the additional hierarchical controllers, and can consolidate nodes across a whole cluster.

The authors in [9], proposed a distributed self-organising approach, where nodes cooperate within the overlay. Migration decisions are performed after an evaluation of the whole overlay state. Similar to our approach, nodes collaborate across the overlay, however the close physical proximity of cooperating nodes in our approach means gossip traffic and migration traffic is localised within the cluster. Similar to us, the authors also concluded that a centralised view is able to achieve better consolidation results, as it has a global view of the infrastructure.

To the best of our knowledge, SHDF is the first hybrid hierarchical distributed framework that enables leaf nodes to operate in a distributed manner, and combines this with hierarchical allocation and consolidation of VMs across large sections of the infrastructure.

## III. Architecture

### A. Framework components

The management of nodes in an IaaS environment can be abstracted as 2 dimensions, *Management Algorithm (MA)* and *Management Framework (MF)*. The MA is responsible for deciding how workloads are assigned to infrastructure resources, while the MF enables the MA to execute by providing common functionality, such as hierarchy level management and aggregation of metrics between nodes. The combined functionality results in workloads executing on infrastructure nodes. In the following sections we briefly describe the MA and detail our proposal, which is in the form of a MF.

### B. Management Algorithm (MA)

In our previous work [2], we have shown management algorithms (MAs) are widely covered in the literature, and drive the decision making process in cloud systems adaptation. Several analytical techniques are used, including Control Theory, Heuristics and Machine learning. Cloud systems adaptation needs to be invoked in order to evaluate the infrastructure and determine whether resource reconfiguration is required. The approaches used in the literature fall onto reactive and proactive engagement. Reactive approaches invoke adaptation at defined time intervals or when a monitored metric, e.g. CPU utilisation, reaches a specific threshold. Proactive approaches predict what demands will be placed on the infrastructure and invoke adaptation ahead of the predicted resource contention point. The MA assigns resources in the infrastructure and regularly assesses the satisfaction of such assignments in achieving a given Service Level Agreement (SLA). The frequency of this assessment is influenced by the time complexity of the algorithm; the lower the complexity, the more frequently the algorithm can be executed. Our proposed architecture enables the MA to run more frequently, and is described in the next section.

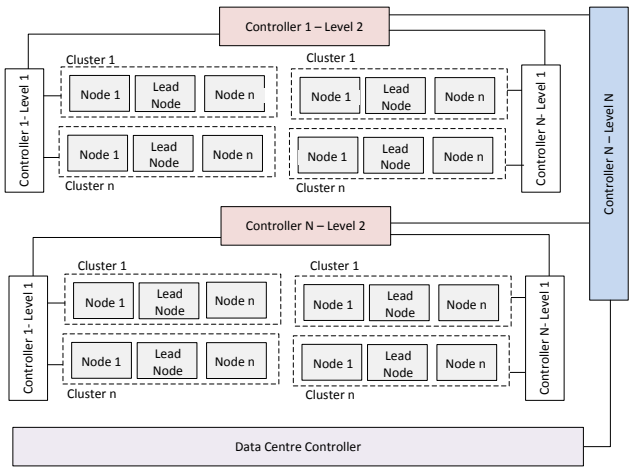



**Fig. 1:** Hierarchical Escalation Architecture

### C. Management Framework (MF)

The MF provides common utilities that enable the MA to execute, including a mechanism to propagate node state, and a decision engine architecture that may be centralised, hierarchical or distributed. The MF is the focus of our work in this paper.

The architecture of our proposed framework SHDF, as shown in Figure 1, consists of hierarchical controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a Node Controller (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. A collection of NCs form a cluster of nodes. Each NC cooperates with a Lead Node (LN), which is a higher level controller for all the NCs within a given cluster. Unique to our proposal, the NCs within each cluster are divided into logical groups, called overlays, where a NC cooperates with other NCs within the same overlay. Each NC exists in only one overlay and in one cluster. This architecture enables nodes to cooperate in a hybrid hierarchical distributed manner.

Unique to our approach, the LN operates as a normal node within the infrastructure in addition to its management responsibility towards the cluster. A collection of clusters creates a level, $n$, which is managed by a controller at level *n+1*. At the highest level, the Data Centre Controller (DC) manages the controllers one level below it.

SHDF attempts to service resource requests at the lowest local level possible, in order to reduce the overhead of servicing the request [22] and to reduce the performance impact of migrating VMs across cluster boundaries [6]. All the controllers in SHDF manage multiple execution nodes, with the exception of the Node Controller (NC), which manages a single node.

The following subsections describe the components of SHDF and the functionality it provides to the MA.

**TABLE I:** Exchanged data between overlay nodes

| Field | Description |
|---|---|
| NodeID | The ID of the node, data belongs to |
| VMSize[NumberAvailable] | number of predefined VM sizes that can be hosted on this node |
| TimeStamp | Time stamp for captured metrics |

*1) Overlay Management:* Execution nodes operate in a distributed way within the boundary of a cluster and are organised in multiple overlays. An overlay is a fundamental concept to reliable multicast, as it abstracts the details of the underlying physical network by building a virtual network on top of it, which can be seen as a graph that represents a link between nodes. To construct overlays, typically two main approaches exist [23]: structured and unstructured protocols. Structured approaches tend to be efficient in terms of number of links, but are sensitive to node failure, because the overlay structure takes into account metrics such as latency or bandwidth, and upon node failure the structure needs to be rebuilt. In unstructured approaches, links are established randomly among the nodes. To ensure overlay stability, multiple links are established to ensure redundancy, however this can cause nodes to receive multiple copies of a given message through its different neighbours.

SHDF is closer to structured approaches [24], where the overlay is constructed to span a subset of a cluster, and by definition is layered on a physical structure of nodes, which solves the proximity challenge in structured approaches, without needing to associate latency metrics with each of the overlay links. When a new node is added to the infrastructure, it gets added to a cluster and assigned an overlay. SHDF enables the MA to determine where a new node is added. When a node is added to an overlay, it receives details of neighbouring nodes from the same overlay, through the data dissemination mechanism described below.

*2) Data dissemination:* In centralised and hierarchical architectures, a central component receives state updates from all the nodes under its management, to enable the MA to adapt the infrastructure. SHDF removes this dependency and enables the nodes to exchange their state within an overlay. Each node holds a local cache of the state of other nodes in the overlay, and to populate this cache, nodes exchange state messages. A node selects another node at random from the overlay and exchanges its view of the overlay. We use a pull & push gossip approach, where node *x* sends new state updates to node *y*, and retrieves new states updates at node *y*. A new node state takes O(log N) rounds to reach all nodes, where N is the number of nodes in the overlay [25]. A gossip round completes when every node has initiated an exchange exactly once. The populated local cache allows the MA to perform VM migration to other nodes within the overlay, and allows the LN to perform VM consolidation across the cluster.

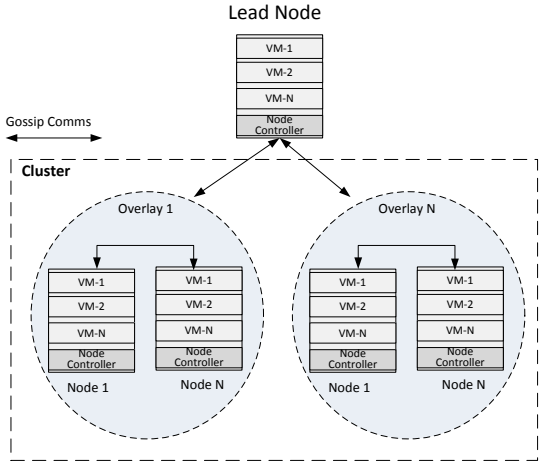



**Fig. 2:** Node Controller Overlays

Similar to [7], a NC regularly exchanges state with other NCs and holds a cache of free capacity of other cooperating controllers in the same overlay, shown in Table I. The gossiping algorithm is shown in Algorithm 1. We enhance this further and make nodes gossip, when there is a state change such as inward migration or a node is being shutdown. Each node will keep the latest metrics seen, and every time an exchange of state occurs the timestamp is used to decide if each exchanged entry is later than any existing entries. If it is, the node's internal state is updated, otherwise the data exchanged is discarded. The MA running on each node can use these metrics to enable nodes to cooperate in resource allocation requests, which will be described later.

**Algorithm 1** stateExchange@Node

```
1:  procedure EXCHANGESTATE(localState)
2:      overlay ← getOverlayMembers - thisNode
3:      randomMember ← random(overlay)
4:      remoteState ← randomMember.gossip(localState)
5:      for i in remoteState do
6:          remote ← remoteState[i].timeStamp
7:          local ← localState[i].timeStamp
8:          if remote > local then
9:              localState[i] ← remoteState[i]
10: procedure GOSSIP(remoteState)
11:     respond to caller ← getLocalSate
12:     receivedState ← remoteState
13:     for i in receivedState do
14:         recived ← receivedState[i].timeStamp
15:         local ← localState[i].timeStamp
16:         if received > local then
17:             localState[i] ← receivedState[i]
```

As in Figure 2, the Lead Node is an ordinary node, it will also exchange its own state. The LN is a member of all the overlays in the cluster and will receive state exchanges from all the nodes within the cluster. To restrict the view of a node and reduce redundancy of state management, when a node gossips with the LN, it will only receive state updates from its overlay and not from other overlays the LN knows about.

SHDF enables the LN to send aggregated metrics for all the nodes within its cluster, to other cooperating controllers. The LN participates in an overlay with other LNs, in a similar way to the NCs, and it can exchange aggregated metric data. The parent controller to the LN is always included in each LN overlay, resulting in state exchanges between the LN in an overlay and the managing controller, in a similar way to the exchanges between a LN and a NC, as shown on Figure 3. The aggregated data can used by controllers at higher levels in the hierarchy, resulting in an aggregated view of the whole infrastructure to be available at the DC controller level.

*3) Controller functionality - VM migration:* When a node cannot satisfy demands of the VMs it hosts, it starts an escalation process that aims to resolve the request at the lowest possible level. The MA running on the stressed node and the LN need to cooperate to resolve the escalated VM migration, by using our provided framework mechanisms.

Table II shows the available methods to support migration and escalation of a VM. SHDF uses different methods to migrate within the overlay and escalate outside it, with a higher priority for escalation. The NC can send a migration request to other nodes within the same overlay, by using the accumulated metrics from other nodes. In turn, the MA running on the LN can query the cluster records from all the overlays, which has state data from all nodes in the cluster, to find a suitable node to house an escalated VM. When there is no free capacity in a single node to match the VM request, but there are enough free resources across the whole cluster, the LN can choose to invoke a consolidation process, where multiple VMs could be migrated between nodes to facilitate creation of enough space to house an escalated VM.

If it is not possible to house the VM escalation within the cluster, the MA running on the LN can use the aggregated metrics, of free capacity, from other LNs and decide to forward the escalation to another LN in the same overlay as the originating LN. If no suitable node is found, the MA can escalate the request to a higher level controller (LN parent) with a broader view of the infrastructure. This is repeated until a node is found or the escalation reaches the DC Controller, which has a view of the entire data centre, and the MA on the DC can choose to direct the request to other parts of the hierarchy, by utilising the aggregated data received by the DC controller. As requests progress through the escalation process, they are assigned an increasing priority, which can be used by the MA in the decision making process. For example the MA may choose to prioritise finding a host for an escalated VM compared to a new VM placement. If the aggregated metrics at the DC show no availability to house the escalated VM, then the request is rejected and the originating node is notified.

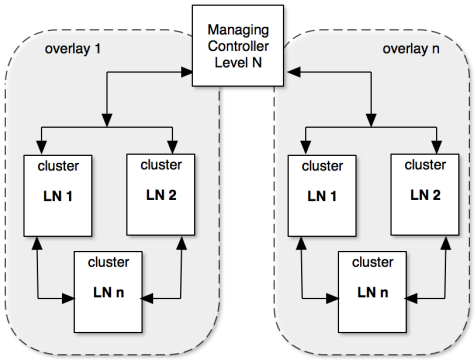



**Fig. 3:** Lead Node Overlays

**TABLE II:** Controller functionality for Migration of VMs

| Method | Source | Destination |
|---|---|---|
| Migrate VM | Node a | Node b |
| Escalate VM | Node a | LN |
| Place Escalated VM | LN | Node b |
| Escalate VM | LN a | LN b |
| Escalate VM | LN a | Level n Controller |
| Escalate VM | Level n Controller | Level n +1 Controller |
| Escalate VM | Level n Controller | DC |
| Place Escalate VM | DC | Level n -1 Controller |
| Place Escalate VM | Level n Controller | Level n -1 Controller |
| Place Escalate VM | Level n Controller | LN |

*4) Controller functionality - Consolidation:* At periodic time intervals and changes in utilisation, each of the management controllers, and LNs, can invoke a consolidation process where the MA can examine the state of the infrastructure and for every node under its management, decide to :

- Migrate some VMs from a node
- Migrate all VMs off a node and switch the node off
- No change

The advantage of SHDF is it allows the nodes to primarily operate in a distributed manner for time sensitive operations such as VM migrations. For non time sensitive operations like VM placement and consolidation, SHDF enables the running of time complex algorithms across parts of the infrastructure or even the whole infrastructure. Compared to distributed approaches, this will enable the MA to have a much more comprehensive view of the infrastructure during these operations.

*5) Bootstrapping & Hierarchy Management:* We defer development of the some of the functionality of SHDF, including ability to bootstrap, dynamically adapt the hierarchy, high availability of the LN and clock synchronisation to further work in the future.

## IV. Implementation of SHDF

In order to evaluate SHDF, we combined it with a Management Algorithm (MA) to utilise the capabilities of our proposed architecture. The following is a description of the MA components. To focus our evaluation on the capabilities of the MF and not the features of the MA, we choose a simple MA from other work in the literature [18], which has been cited by other researchers, and uses a threshold based approach to engage adaptation of the infrastructure.

SHDF design enables several levels of hierarchy, with controlling managers at each level. To simplify initial experiments, we choose the smallest number of levels, and instantiated SHDF with 3 levels of management with NCs at Level 1, LNs at Level 2 and a Data Centre controller at Level 3. We intend to implement and evaluate more levels in the future.

### A. VM Migration

As in Algorithm 2, we use the utilisation metrics available to the NC and send a migration request to candidates in the same overlay, marking them as tried, and wait for a response. The NC sorts candidate targets in the following order: Partially utilised (<upper threshold), under utilised (<low threshold) and empty (0 VMs), and tries them in this order, to achieve higher node utilisation and reduce selection of powered off nodes.

**Algorithm 2** Create VM Migration@Node

```
1: x ← chooseVMConsumingMostResource()
2: sort(PartialUtilisation, lowUtilisation, empty)
3: for k in nodeNeighborList do
4:     if k canHost x AND notTried k then
5:         send migrationReuest to k
6:         mark k tried
7:         return
8: EscalateToLeadNode()
```

Once a candidate is marked as tried, we do not use it again until we see an updated state from it through the gossiping mechanism.

When the LN receives the escalated request, as in Algorithm 3, it will be able to forward the request to all eligible nodes within the cluster. As this is an escalated request and it came from within an overlay within the cluster, the sending overlay is excluded from the list of eligible target nodes.

**Algorithm 3** Process VM Migration@LN

```
1: req ← migrationRequest
2: nodes ← clusterNodes - sendingOverlayNodes
3: sort(nodes, PartialUtilisation, lowUtilisation, empty)
4: for k in nodes do
5:     if k canHost req AND notTried k then
6:         send req to k
7:         mark k tried
8:         return
9: EsclateToDC
```

When a node gets an escalated migration request, via the LN, it processes it in a similar way to requests from within the overlay. If it can be accommodated then an accept request is sent to the original sender, otherwise a reject request is sent to the LN. The LN will keep forwarding the migration request to nodes within its cluster until it has tried all nodes, and if it fails to place the migration, it will escalate the request to the DC controller, one level above it. The DC Controller will search for a cluster that can host the VM based on aggregated metrics from all the LNs.

A DC Controller selects the first target cluster that can host the escalation, and forwards the request to the LN responsible for that cluster. When the identified LN receives the escalated migration request it processes it in a similar way to escalation from within its own cluster, in that it identifies candidate nodes and forwards them the request. If the receiving LN cannot host the migration within its cluster it replies with a rejection to the DC, which in turn will attempt other clusters until it exhausts all clusters. At this point we have exhausted the entire infrastructure and failed to find a suitable node to house the VM migration. The DC will then send a final reject message to the original migrating node.

### B. VM Placement & Conslidation

We implement the VM placement and consolidation approach in [18], and utilise the SHDF architecture to disseminate metrics to the LN and DC.

## V. Experiment setup and Evaluation

We use simulation to facilitate rapid development of experiments of large data centres. We selected DCSim [26] because of its extensibility and existing implementations for hierarchical and distributed approaches, allowing us to create baseline comparisons for our proposed MF. To focus our evaluation on the features of the architecture and not the features of the MA, we implemented the MA from [18] with SHDF. We also utilised the existing proposals from [18] for the hierarchical and [27] for the distributed baseline comparisons. Source code for both approaches is readily available and has been cited by other researchers. The baseline MAs are similar, and are based on a greedy heuristic, that reacts to utilisation thresholds. While the MA used in the comparison does not include several features, such as workload prediction and migration cost, using the same MA allows us to assess the capability of the MFs.

We instantiate both SHDF and the hierarchical approach [18] with 3 levels of controllers, running on the root of the data centre, the cluster manager and leaf nodes.

### A. Simulator Setup

DCSim allows a VM to use more CPU than reserved, up to an amount that does not impact other VMs. Like [18], we use a CPU utilisation thresholds of 90% for high, indicating stress level, and we use 60% for low, indicating low utilisation.

In DCSim, an application is modelled as an interactive multi-tiered web application. Each application has a specified client think time, and a workload component. The workload defines the current number of clients connected to the application, which can change at discrete points in the simulation based on a *trace* file. The resource requirements are defined as its resource size, which is the expected amount of: CPU, memory, bandwidth and storage. DCSim treats bandwidth and storage as fixed requirements, however, CPU requirements can be varied across the simulation based on the VM demands. The DCSim energy model defines how much energy the node consumes at different CPU utilisation levels, and is calculated using results from the SPECPower benchmark. The benchmark provides energy consumption levels of real servers in 10% CPU utilisation intervals, and DCSim uses these values and calculates intermediary values using linear interpolation [26]. We assume that CPU energy consumption is the most contributing factor to a node's energy consumption [28] and we use the default DCSim energy model in our experiments. DCSim applies a cost to VM migration including the time taken for migration, as a function of memory consumed by a VM, and factors the bandwidth required for the VM migration on the hosting node. Additionally, the boot time of a switched off node has an elapsed time cost. The time taken to switch on a node and migration is reflected on the time period the VM is in a stressed state, and therefore the SLA achieved by a VM.

For the gossip protocol, the exchange frequency and fan out values influence propagation speed and reliability. The fan out value determines the number of times a new state is exchanged by a node. For example, when a node receives an update and the fan out value is 2, the node will exchange this newly received state in the next 2 cycles of the gossip protocol. We trade bandwidth consumption with propagation speed, and chose a gossip frequency of 2 minutes and fan out value of 2. Due to limited space, we omitted our analysis of these parameters.

*1) Workload & SLA:* We run the experiments at a load that requires on average 80% of the CPU resources of the data centre. Each simulated application contains a workload trace based on the number of incoming requests to web servers from publicly available traces. Due to limited space, we limit our investigation to the `google_corest_job_type` included with DCSim and create VMs that demand 4 cores, 4.5GHz core capacity and 2GB RAM. DCSim is able to track SLA violations based on the response time of VMs, and we set this to trigger any time the response time is above 1 second.

*2) Data centre:* Our experiments use nodes modelled as ProLiant DL160G5 [29], with 2 quad-core 2.5GHz CPUs and 32GB of memory. The number of nodes used is specified in each of the experiments, and is typically up to 50,000 nodes. We assume that the data centre supports live VM migration, as this technique is currently supported by most major hypervisor technologies, such VMware [30] or Xen [31].

The various parameters used in our evaluation are outlined in Table III.

**TABLE III:** Evaluation setup

| **Variable** | **Value** |
|---|---|
| Low threshold value (triggers consolidation) | 60% CPU utilisation [18], [27] |
| High threshold value (triggers migration) | 90% CPU utilisation [18], [27] |
| Node Monitoring Period | 2 minutes [18], [27] |
| Consolidation Frequency | Every 1 hour |
| Size of Overlay | 100 nodes |
| Gossip Frequency | 2 minutes |
| Fan out value | 2 |
| Delay before node is switched on, from Off State (impacts SLA) | 3min [18], [27] |
| SLA Threshold | 1 Second [18], [27] |
| Simulation Duration | 3 Days |

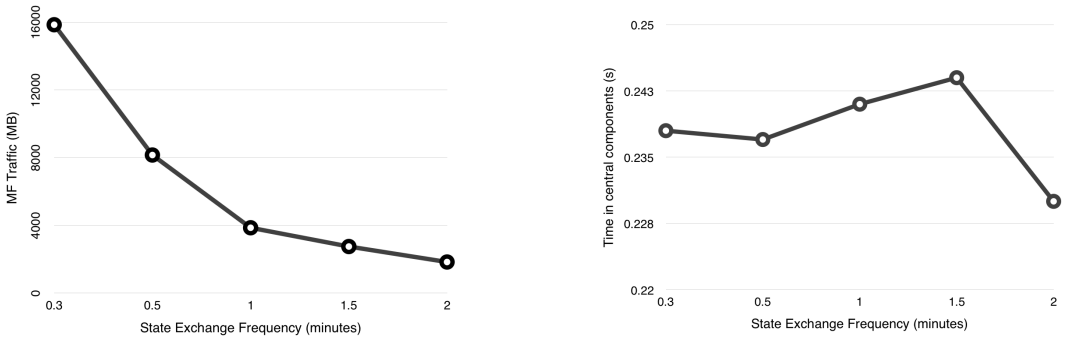
### *B. Evaluation: SHDF settings*

The frequency at which the gossip protocol executes influences the speed of data dissemination. Figure 4 shows the impact of running the gossip protocol at different time intervals on 5 clusters of 1000 cooperating nodes, 5000 total. The more frequently the protocol runs, the higher the traffic consumption, as shown by figure 4a, which is the trade off between delivering fresh metrics and traffic consumption. Any MA combined with SHDF can configure the frequency of this setting. Encouragingly the scalability of central components remained consistent (milliseconds variation) as the protocol executed more frequently, as shown in Figure 4b. This shows SHDF has a low overhead as MF.

We have also run experiments with different parameters for *size of Overlay and fan out values*. Larger overlay sizes consume larger amount of MF traffic, and the scalability remained constant. Increasing the fan out value also increased the amount of consumed management traffic. We have omitted these results due to limited space.

### *C. Evaluation: Scalability*

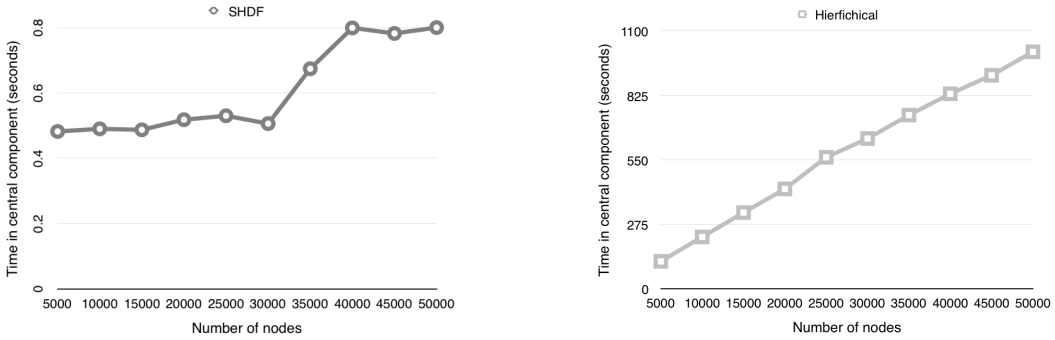
Scalability bottlenecks are typically found in centralised access to resources, with distributed systems improving scalability by parallelising computation and resource access. To evaluate the scalability of both the hierarchical approach and SHDF, we measure the time cost of central components on both approaches. Distributed approaches typically do not have a central migration component and thus are not relevant in this comparison. For the hierarchical approach, the central migration components are: cluster metric aggregation, node stress detection and VM escalation. For SHDF, the central migration components are: cluster metric aggregation and VM escalation. As SHDF is using the same MA as the hierarchical approach, examining the time in central components will allow us to compare the features of the MF. Figure 5a shows SHDF spends milliseconds in central components, compared to 100s of seconds by the hierarchical MF, resulting in an inherently scalable architecture. The hierarchical approach on the other hand, had a near linear increase in central components, suggesting a high fixed cost as the number of nodes increase, which is is due to the central processing of metrics and actioning migration on behalf of every stressed




**(a)** SHDF Traffic consumption

**(b)** Impact on scalability of central components

**Fig. 4:** SHDF State Exchange - 5000 nodes




**(a)** SHDF time cost

**(b)** Hierarchical FM time cost

**Fig. 5:** SHDF versus Hierarchical Scalability

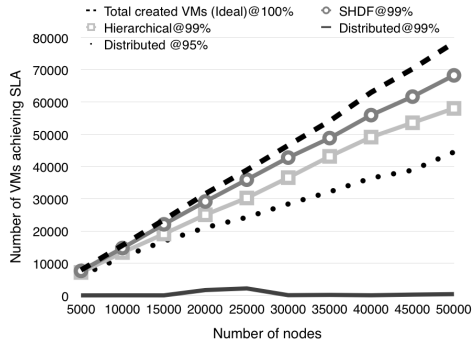
node. For the largest experiment of 50,000 nodes, SHDF was configured with a NC in each node and 50 LNs, each managing a cluster of 1000 nodes. Both NCs and LNs are execution nodes and only the DC Controller is modelled on a dedicated management node. In contrast to the framework in [17], where nodes are either execution or management nodes.

The increase in time cost between 30,000 nodes and 40,000 in 5a is due to how the MA [18], used by SHDF, finds available nodes during an escalation. This can be optimised by periodically sorting and caching the available nodes at the LN, which will reduce the search time.

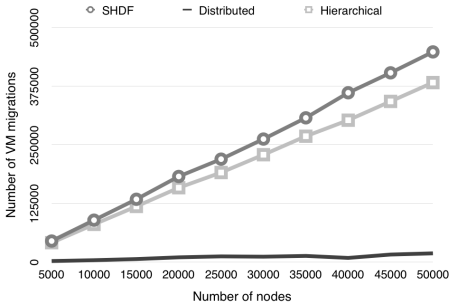
The MA [18] implemented by both the hierarchical approach and SHDF uses a greedy heuristic to find a migration target, and many alternatives in the literature are more complex. The authors in [32] incorporated the cost of migration and [11] incorporated power cost of decision engine in the migration. Additionally, proactive approaches [33], [11] start the VM migration before the conflict occurs, and in [11] the authors proposed performing a cost and benefit analysis before applying migration, and only invoked a migration if the benefit outweighed the cost of the migration. All of these methods to improve the quality of the decision making increase the sophistication of the MA, and combining these techniques with our proposed MF, should result in a highly scalable architecture, and will allow the MA to be deployed in a scalable framework, even when used to manage a large data centre.

### *D. Evaluation: Management Framework Traffic*

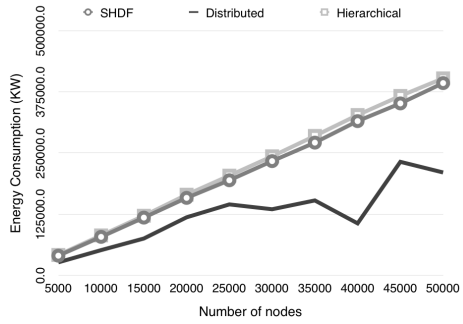
In all of the evaluated approaches, each MF propigates node status to allow the allocation of resources. In the Distributed approach, node status is sent when one of the nodes is stressed and is looking for a migration target and when nodes are cooperating to perform consolidation. This status propagation occurs across the whole data centre, and as shown in Figure 6d, the distributed approach consumes the most management traffic out of all three approaches. In the hierarchical approach, leaf nodes send their status to one higher level controller and do not share their status with other nodes, which results in the least management traffic, at the cost of total reliance on a central component for decision making. In SHDF, nodes exchange their status within the same overlay, to enable rapid cooperation and decision making between the nodes. A node gossips its state at regular intervals and when a VM is migrated into or off a node. Thus during high load scenarios, as in this experiment, SHDF will consume more management traffic in order to propagate status state within the overlay.



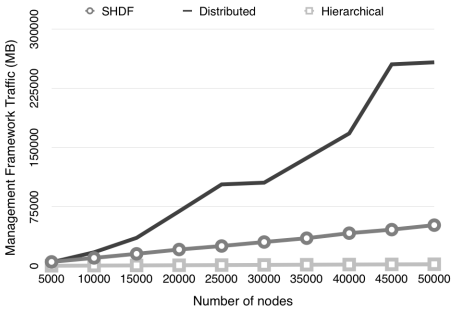

**(a)** SLA Achievement




**(b)** Number of Migrations




**(c)** Energy Consumption




**(d)** Management Framework Traffic

**Fig. 6:** Hierarchical versus SHDF versus Distributed

### *E. Evaluation: SLA & Energy Usage*

Critical to the effectiveness of a MF is to enable the MA to achieve SLAs and balance this with energy consumption of switched on nodes. Using end to end metrics like SLA and energy consumption, allows us to evaluate the influence of the MF, when combined with the same MA.

Figure 6a shows the total number of created VMs, the ideal case at 100% achievement, and the achieved SLA level for the three approaches. Using a similar MA to the hierarchical and distributed approach, SHDF meets more SLA targets than both the hierarchical and distributed approaches, as nodes perform their own stress detection and have a local view of the overlay, which enables each node to make rapid decisions about migrating VMs to an available target node. In the distributed approach, each node also does its own stress detection, but nodes do not have a local cache of the state of other nodes in the data centre. When stress is detected, a node broadcasts a request to migrate a VM away and waits for and then processes responses, resulting in VMs waiting for longer before they are migrated away from stressed nodes, with most VMs not achieving a 99% SLA. Even at 95% SLA achievement level, the distributed approach remains impacted by the lack of rapid decision making, compared to SHDF and hierarchical approaches, and many VMs do not achieve their SLA.

The number of migrations carried out by all of the approaches, Figure 6b, highly correlates to the VM SLA achievement, Figure 6a. Both the hierarchical approach and SHDF migrated VMs several times in order to reduce SLA violations, with approximately 6 migrations per placed VM. The reason for this, is the MA used by both the hierarchical approach and SHDF is a simple heuristic that does not predict workload patterns and engages adaptation reactively, and can cause system oscillation through large number of migrations. In the distributed implementation, once a migration request is broadcasted, the requesting node has to wait for and process the responses. This results in an elapsed time, which reduces the number of executed migrations and the achieved SLA. The hierarchical approach consolidates VMs at the rack level. It will search for under utilised nodes, move their VMs to other nodes in the rack, and switch the under utilised nodes off. SHDF performs a similar role across the cluster level, and as a cluster typically contains several racks, SHDF has a higher number of target nodes. We see the impact of this in Figure 6c, where both approaches consume a similar amount of energy, with SHDF being able to keep nodes switched off for longer. The distributed approach is not able to achieve VM demands quickly enough and thus incurred SLA violations, and in turn it did not consume as much energy as the other approaches.

## VI. Future work and Conclusion

We presented SHDF for managing cloud data centres, in which the nodes of the data centre are physically organised in the typical cluster formation and leaf nodes are then able to cooperate within logical boundaries of an overlay.

Additionally, the framework allows this pattern to be repeated at higher levels in the hierarchy. SHDF enables a MA to solve the allocation of resources and to configure the attributes of SHDF. Evaluation of SHDF shows its ability to retain the properties of a chosen MA, while increasing scalability by reducing time spent in centralised components. Further areas of research include adding adaptability to the framework to allow it to adjust and fine tune its parameters dynamically. For example, in a high load scenario, it might be more optimal to increase the propagation of the gossip exchange to allow nodes to have a fresher view of nodes in the overlay. In contrast, in low load scenarios, nodes can conserve bandwidth by lowering the frequency of gossip exchanges.

Further steps are necessary to investigate the impact of node failure on our gossip protocol, and its ability to continue propagating state changes within the overlay.

## REFERENCES


[1] R. Miller. Data center knowledge. [Online]. Available: http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/

[2] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Adaptation in cloud resource configuration: a survey," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1–16, 2016.

[3] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *Autonomic Computing ICAC*. Washington, DC, USA: IEEE, Jun 2008, pp. 3–23.

[4] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang, "Autonomic management of cloud service centers with availability guarantees," in *2010 IEEE 3rd International Conference on Cloud Computing*. Washington, DC, USA: IEEE, July 2010, pp. 220–227.

[5] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, pp. 253–272, 2013.

[6] A. Aldhalaan and D. A. Menascé, "Autonomic allocation of communicating virtual machines in hierarchical cloud data centers," in *Cloud and Autonomic Computing (ICCAC), 2014 International Conference on*, Sept 2014, pp. 161–171.

[7] F. Wuhib, R. Yanggratoke, and R. Stadler, "Allocating compute and network resources under management objectives in large-scale clouds," *Journal of Network and Systems Management*, vol. 23, no. 1, pp. 111–136, 2015.

[8] M. Sedaghat, F. Hernández-Rodriguez, E. Elmroth, and S. Girdzijauskas, "Divide the task, multiply the outcome: Cooperative vm consolidation," in *IEEE International Conference on Cloud Computing Technology and Science*. Washington, DC, USA: IEEE, Aug 2014, pp. 300–305.

[9] D. Loreti and A. Ciampolini, "A decentralized approach for virtual infrastructure management in cloud," *International Journal on Advances in Intelligent Systems*, vol. 7, no. 3/4, pp. 507–518, 2014.

[10] N. M. Calcavecchia, B. A. Caprarescu, E. Di Nitto, D. J. Dubois, and D. Petcu, "Depas: a decentralized probabilistic algorithm for auto-scaling," *Computing*, vol. 94, no. 8, pp. 701–730, 2012.

[11] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE, 2010, pp. 62–73.

[12] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: Integrated capacity and workload management for the next generation data center," in *International Conference on Autonomic Computing*. Washington, DC, USA: IEEE, Jun 2008, pp. 172–181.

[13] J. Almeida, V. Almeida, D. Ardagna, Í. Cunha, C. Francalanci, and M. Trubian, "Joint admission control and resource allocation in virtualized servers," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 344–362, Apr 2010.

[14] H. Moens, J. Famaey, S. Latre, B. Dhoedt, and F. D. Turck, "Design and evaluation of a hierarchical application placement algorithm in large scale clouds," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2011, pp. 137–144.

[15] H. Goudarzi and M. Pedram, "Hierarchical sla-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 222 – 236, June 2016.

[16] O. Mola and M. Bauer, "Towards cloud management by autonomic manager collaboration," *International Journal of Communications, Network and System Sciences*, vol. 4, no. 12A, pp. 790–802, 2011.

[17] H. Moens and F. D. Turck, "A scalable approach for structuring large-scale hierarchical cloud management systems," in *9th International Conference on Network and Service Management (CNSM)*, 2013, pp. 1–8.

[18] G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "A hierarchical, topology-aware approach to dynamic data centre management," in *Network Operations and Management Symposium (NOMS)*, 2014, pp. 1 –7.

[19] H. N. Van, F. D. Tran, and J.-M. Menaud, "Sla-aware virtual resource management for cloud infrastructures," in *IEEE International Conference on Computer and Information Technology*, vol. 02. Washington, DC, USA: IEEE, 2009, pp. 357–362.

[20] F. Wuhib, R. Stadler, and M. Spreitzer, "Dynamic resource allocation with management objectives: implementation for an openstack cloud," *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 213–225, 2012.

[21] M. Pantazoglou, G. Tzortzakis, and A. Delis, "Decentralized and energy-efficient workload management in enterprise clouds," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 196–209, April 2016.

[22] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for cloud infrastructure management," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 472–487, Feb 2013.

[23] M. Matos, A. Sousa, J. Pereira, R. Oliveira, E. Deliot, and P. Murray, *CLON: Overlay Networks and Gossip Protocols for Cloud Environments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 549–566.

[24] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Hiscamp: Self-organizing hierarchical membership protocol," in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, ser. EW 10. New York, NY, USA: ACM, 2002, pp. 133–139.

[25] K. Birman, "The promise, and limitations, of gossip protocols," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 8–13, Oct. 2007.

[26] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, 2012, pp. 385–392.

[27] M. Tighe, G. Keller, M. Bauer, and Lutfiyya, "A distributed approach to dynamic vm management," in *Proceedings of the 9th International Conference on Network and Service Management*, 2013, p. 166 to 170.

[28] A. Beloglazov, J. Abawajyb, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, pp. 755–768, May 2012.

[29] Hpe proliant. [Online]. Available: https://www.hpe.com/uk/en/product-catalog/servers/proliant-servers.html

[30] Vmware. [Online]. Available: http://www.vmware.com/

[31] Citrix. Xen. [Online]. Available: http://www.xenserver.org

[32] M. Sedaghat, F. Hernández-Rodriguez, and E. Elmroth, "Autonomic resource allocation for cloud data centers: A peer to peer approach," in *IEEE International Conference on Cloud and Autonomic Computing*. Washington, DC, USA: IEEE, Sep 2014, pp. 131–140.

[33] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 5:1–5:14.

## 4.3 A Hierarchical Decentralized Architecture to enable Adaptive Scalable Virtual Machine Migration

Abdul R Hummaida, Norman W Paton and Rizos Sakellariou

**Publishing state:** submitted to Concurrency and Computation: Practice and Experience (CCPE), 2021.

**Summary**: To evaluate further the proposed hybrid hierarchical decentralized management framework (MF), we assess and compare the performance of multiple management algorithms (MAs) from the literature. We also compare the MAs' performance with hierarchical, decentralized and centralized MFs. When running on the hybrid MF, all MAs retain their SLA performance properties, and some MAs exhibit higher SLA performance due to a reduced search space and autonomous properties of the Hybrid MF. This demonstrates the feasibility to separate the MF and MA, the flexibility of the hybrid MF, and the ability to integrate it with other MAs to investigate cloud resource management.

Each experiment is run to simulate 24 hours of elapsed time and each simulated application contains a workload trace from the public traces included in DCSim. The simulation is designed to instantiate a certain number of VMs per hour, where each VM will run one of the public traces in a round-robin approach. For example, when 1000 VMs are created per hour and 5 traces are used, 200 VMs will use each of the traces per hour. Each trace sets the size of incoming traffic to a VM and typically changes every 5 minutes. As the round-robin approach is used in the experiments for each of the MA and MF evaluations, we assume this is a valid method to instantiate the VMs.

The hybrid architecture includes a virtual layout of nodes, which does not affect the physical layout of the infrastructure. The construction of virtual overlays is envisaged as a process through an administrator portal, where the size of overlays would be specified. This can be used in the initial instantiation of the infrastructure or as an adjustment of an existing setup. In the latter case, each node will be supplied with a new list of nodes to cooperate with and the rest of the processes will remain the same. In terms of choosing a size of the overlays, our experiments show that the benefits of the hybrid MF can be achieved from overlays as small as 25 nodes, with our default experiments using 50 nodes in an overlay. The ability to configure different sizing

enables the cloud operator to match the overlay configuration to mimic their physical infrastructure. This is a simple configuration that does not have a significant impact on the operation of the proposed approach.

To reduce the variables in the evaluation of the different approaches, we use a first-fit heuristic to perform the initial VM placement in all of our experiments.

**Key contributions:** Contribution 3 (see Section 1.4).

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

# A Hierarchical Decentralized Architecture to enable Adaptive Scalable Virtual Machine Migration

Abdul R Hummaida* | Norman W Paton | Rizos Sakellariou

[1]University of Manchester, Department of Computer Science, Kilburn Building, Oxford Rd, Manchester M13 9PL, UK

**Correspondence**
*Corresponding author: Abdul R Hummaida
Email: abdul.hummaida@postgrad.manchester.ac.uk

**Summary**

Cloud infrastructure is an established mechanism for end users to access flexible resources. Infrastructure providers seek to maximise accepted requests, meet Service Level Agreements, and reduce operational costs by dynamically allocating Virtual Machines to physical nodes. Many solutions have been presented to manage cloud infrastructure, however, these tend to be centralized and suffer in their ability to maintain Quality of Service (QOS) and support data centres with thousands of nodes. Decentralized approaches, with no central management, can manage large data centres. However, these tend to reduce the ability to an optimal resource allocation across the data centre. To address this, we propose a hybrid hierarchical decentralized architecture that achieves lower SLA violations and lowers network traffic. We used simulation to evaluate and demonstrate our proposal in practice with a variety of existing VM placement policies.

**KEYWORDS:**
Datacentre Scalability, Datacentre QOS, VM Migration, Resource Management, Hierarchical architecture, Decentralized architecture

## 1 | INTRODUCTION

Cloud computing environments enable end users to access to computing resources through a simplified a service model, instead of purchasing, configuring and maintaining these resources. Infrastructure Providers (IPs) typically abstract data centre resources and present them to customers through a virtualisation layer, with a Virtual Machine (VM) as a common form, with each VM being isolated from other VMs. End users of cloud resources typically request these through web APIs, which map the user's requests to virtual resources that reside on physical resources in the datacenter[1].

VM Placement involves efficient utilisation of available resources for applications to meet performance goals such as SLA. To place VMs, a scheduling process needs information on the resource requirements of the VMs being allocated, such as CPU and number of cores, amount of memory and network bandwidth. Once a resource is provisioned, IPs track the varying demands on each resource, and this varies over time as end users consume and release these resources.

To meet workload demands, IPs may reconfigure the assignment of physical resources to VMs, by increasing or reducing resource allocation to a workload, through Elasticity[2]. Resource reconfiguration is complex and may cause service interruption, hardware wear and tear and system instability[3]. VM migration is a common reconfiguration case and has been applied at scale[4]. Proposals in the literature aim to minimise Service Level Agreement (SLA) violations, and some trade this off with another objective, such as reducing energy consumption or maximising IP revenue and typically focus on the CPU as the primary resource to manage[5]. Idle nodes consume about 70% of their peak power[6] and approach focusing on minimising energy consumption

attempt to achieve high CPU utilisation. There are also technologies, such as containerisation, which support the fast deployment of cloud applications. A container housing an application is allocated to a VM, and a VM is allocated to a node[7,8?,9,10]. While container resource management overlaps with our work, we focus on the management and allocation of VMs to nodes.

Centralized, hierarchical and decentralized architectures have their merits. Centralized architectures have a global view, hierarchical have increased scalability compared to centralized, and decentralized architectures have no central controller and can scale to manage a large number of nodes. We hypothesise that the strengths of these architectures can be combined, and their weaknesses reduced. We propose a Scalable Hierarchical Decentralized Framework (SHDF), which overcomes the weaknesses of hierarchical, decentralized and centralized approaches, by achieving improved QoS metrics. SHDF consists of hierarchical controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a *Node Controller* (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. Each NC operates in a decentralized architecture and cooperates with other NCs and a *Lead Node* (LN), which is a higher level controller for all the NCs within a given cluster.

In this paper we extend a preliminary version of this paper[11], by implementing and evaluating QOS metrics on multiple policies from the literature. Our goal is to demonstrate improved QOS metrics compared to centralized, hierarchical and decentralized architectures.

The scalability of a system is defined as the ability to meet a larger workload requirement by adding a proportional amount of resources, and maintain its performance[12]. By achieving improved QOS performance metrics, our proposal should scale better compared to the examined architectures; the extent to which this is the case is explored through empirical evaluation.

We used several of the metrics in the literature[13,14] to evaluate our approach. The objective is chiefly to achieve:

1. Fewer SLA violations compared to centralized, hierarchical and decentralized architectures.
2. Lower network traffic utilisation to manage resources, compared to centralized architectures.
3. Improved scalability, as the number of nodes in the data centre increases, compared to centralized, hierarchical and decentralized architectures.

The rest of this paper is organised as follows. Section 2 describes related work. Section 3 describes the architecture of our proposed scalable framework. Section 4 presents multiple management algorithms and an evaluation of our implementation and compares it to centralized, hierarchical and decentralized architectures. In Section 5 we conclude and discuss future work.

## 2 | RELATED WORK

Infrastructure providers typically build the infrastructure and offer access to virtual resources and managed services[15,16]. VMs reside on physical nodes of heterogeneous capabilities where the performance characteristics of compute, storage and network vary. Adaptation can be applied to reconfigure the data centre to optimise a business objective. Centralized architectures use an engine with a global view of the entire managed infrastructure, and can adapt resources across the entire infrastructure. Hierarchical architectures typically divide the infrastructure into multiple clusters, with a decision engine in each cluster[17]. Decentralized architectures distribute the management of the infrastructure without a centralized controller. Both hierarchical and decentralized architectures are a form of distributed management architectures.

As the aim of our work is to investigate the architecture of the decision making process, we categorise related work based on the management architecture used. In particular, we focus on hierarchical and decentralized architectures as these make the constituent parts of our proposal.

### 2.1 | Hierarchical Controllers

Hierarchical approaches, a type of distributed decision making, have been widely studied in cloud resource management. The approaches distribute the decision making process and typically use a multi level hierarchical approach running at different time intervals. In[17] the lowest level controller runs every hour and performs VM placement, power management and workload profiling and was evaluated to 7,200 nodes. In[18], the lowest level controllers manage a small number of machines and the applications that are hosted on them. At the next higher level, a controller manages machines owned by multiple lower level controllers. In[19] 3 levels are used, where the highest level controller managed multiple clusters operating at seconds (L1), minutes

(L2) and days (L3) intervals, however, the scalability of the approach was not explored. In[20] the hierarchy is sliced along the operations of the controllers. A Level 1 controller handles VM placement and load balancing and runs every 30 minutes. A Level 2 controller handles the resources of a node and runs every few minutes. In[21,22] VM placement is performed using 2 level hierarchical controllers, and in[22] a local controller guarantees stability and performance of an application. The global controller models a set of operating points from the local level to address load balancing and achieve a given objective.

Hierarchical proposals typically utilise a controller running in a centralized manner within the scope of a cluster of nodes. As the number of nodes within the cluster increases, the decision making algorithm faces the same challenges as traditional centralized approaches, with the execution time resulting in an inability to react to SLA violations. Our proposed framework tackles this by reducing the operations performed by the centralized controllers and moving management functionality to lower execution nodes, therefore providing a hybrid hierarchical decentralized framework. In our approach, each node controller autonomously manages the execution node, including detection of stress states and violation of SLAs. These node controllers can cooperate with other nodes and higher level controllers to perform management of the infrastructure and workloads.

In[23] a hierarchical approach with VM migration escalation is used. The approach performs the initial assignment of VMs to clusters (containers) and periodically, lower controllers decide what to optimise and pass the decision to parent controllers. The approach considers the time complexity of the decision making algorithm and places an upper bound on optimisations to be performed by the hierarchical controllers.

Our approach has similarities to[24], which proposed a framework for managing a hierarchical cloud management system. Due to the computational requirements of management nodes, the approach proposes every node is either used as a management node or an execution node, never playing both roles. This contrasts with our approach, in which execution nodes can also perform management roles, resulting in fewer nodes dedicated to the management of the infrastructure. Our approach additionally enables the leaf nodes to operate in a decentralized manner. Proposals typically do not build cooperation between controllers, resulting in controllers operating independently. In contrast,[19,25,26,27] proposed controller cooperation. In[19,26], the lower level controllers propagated workload satisfaction with assigned resources to a higher controller, which is then used to possibly further optimise resource assignment. In[25], the cooperating controllers, where lower level controllers can escalate a request to a higher controller, rather than waiting for action in the next management cycle. However, escalation only occurred from a mid level controller within the hierarchy and did not escalate from the lowest level controllers, which our approach does. Scalability was typically not evaluated in these approaches.

Mesos[28] uses a two-level scheduling mechanism, with a master process managing slave daemons running on cluster nodes. User frameworks that run tasks execute on the slave nodes. The master process makes resource offers to User frameworks, which they can accept or reject. Mesos was evaluated to manage 50,000 nodes. However, user frameworks need to be modified to be Mesos aware.

## 2.2 | Decentralized Controllers

In[29,30], a heuristic as a peer-to-peer protocol, enables nodes to communicate directly without a centralized controller. Two cooperating nodes determine whether to migrate a VM based on the defined objectives. While[29] did not take into account the cost or duration of the conflict before applying the migration,[30] incorporated migration cost into the decision making. A periodic node discovery service enables nodes to find new neighbouring nodes to communicate with. On each round of the protocol, two cooperating nodes determine to migrate a VM based on defined objectives. Using simulation, the approach claims to manage more than 100,000 nodes. A challenge with decentralized approaches is the lack of a global view of the infrastructure, which impacts the ability to reach a globally optimal solution. In contrast, our approach has the scalability characteristic of decentralized systems and a global view of the infrastructure through controllers at each level of the hierarchy.

A distributed probabilistic algorithm was used in[31] and utilised an overlay network and a decentralized load balancing technique. A decentralized approach is proposed in[32] for managing the workload of large, enterprise cloud data, focusing on reducing energy consumption and SLA violations. The approach uses a *hypercube* structured overlay, with similar cost to our approach, with N nodes reaching status propagation in O(log N) rounds.

A decentralized approach is proposed in[33], where nodes broadcast resource requests to all nodes during migration. In contrast, our approach has a bigger view of the infrastructure, through the additional hierarchical controllers, and can consolidate nodes across a whole cluster.

A decentralized self-organising approach is proposed in[34], where nodes cooperate within the overlay. Migration decisions are performed after an evaluation of the whole overlay state. Similar to our approach, nodes collaborate across the overlay, however,

the close physical proximity of cooperating nodes in our approach means gossip traffic and migration traffic is localised within the cluster. Similar to us, the authors also concluded that a centralized view is able to achieve better consolidation results, as it has a global view of the infrastructure.

A decentralized reinforcement learning-based controller proposed in [35,36]. Each autonomous node is responsible for managing the performance of its own hosted applications, and collaborates with other nodes. Each node has two monitoring components for keeping track of the applications and the system resources respectively, and a learning-based controller. However, scalability of managing a large infrastructure was not evaluated.

To the best of our knowledge, SHDF is the first hybrid hierarchical decentralized framework that enables *leaf nodes to operate in a decentralized manner, combines this with hierarchical escalation of migration and consolidation of VMs across large sections of the infrastructure*. This utilises the benefits of scale in decentralized systems and global view of hierarchical systems.

## 3 | HYBRID ARCHITECTURE

The management of infrastructure resources and applying Infrastructure Providers (IPs) objectives is a complex challenge. We classify the management process into 2 dimensions, *Management Algorithm (MA)* and *Management Framework (MF)*. The MA is responsible for deciding how workloads are assigned to infrastructure resources, while the MF enables the MA to execute by providing common functionality, such as hierarchy level management and aggregation of metrics between nodes. The combined functionality results in workloads executing on infrastructure nodes. In the following sections we briefly describe the MA and detail our proposal, which is in the form of a MF.

### 3.1 | Management Algorithm (MA)

In our previous work [5], we have shown management algorithms (MAs) are widely covered in the literature, and drive the decision making process in cloud systems adaptation. Several analytical techniques are used, including Control Theory, Heuristics and Machine Learning. Cloud systems adaptation needs to be invoked in order to evaluate the infrastructure and determine whether resource reconfiguration is required. The approaches used in the literature fall into reactive and proactive engagement. Reactive approaches invoke adaptation at defined time intervals or when a monitored metric, e.g. CPU utilisation, reaches a specific threshold. Proactive approaches predict what demands will be placed on the infrastructure and invoke adaptation ahead of the predicted resource contention point. The MA assigns resources in the infrastructure and regularly assesses the satisfaction of such assignments in achieving a given Service Level Agreement (SLA). The frequency of this assessment is influenced by the time complexity of the algorithm; the lower the complexity, the more frequently the algorithm can be executed. Our proposed architecture enables the MA to run more frequently, which opens the opportunity to apply adaptation of the infrastructure more frequently, and could lead to a more optimal assignment of resources.

### 3.2 | Management Framework (MF)

The Management Framework (MF) provides common utilities that enable the MA to execute, including a mechanism to propagate node state, and a decision engine architecture that may be centralized, hierarchical or decentralized. The MF is the focus of our work in this paper.

The architecture of our proposed framework SHDF, as shown in Figure 1, consists of hybrid hierarchical decentralized controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a Node Controller (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. A collection of NCs form a cluster of nodes. Each NC cooperates with a Lead Node (LN), which is a higher level controller for all the NCs within a given cluster. Unique to our proposal, the NCs within each cluster are divided into logical groups, called overlays, where a NC cooperates with other NCs within the same overlay. Each NC exists in only one overlay and in one cluster. The size and method for constructing an overlay is configurable. This hybrid hierarchical decentralized approach enables nodes to cooperate, and thus utilise the benefit of decentralized architectures. Additionally, the hierarchical element of our proposal enables cluster wide view for consolidation of VMs.

Unique to our approach, the LN operates as a normal node within the infrastructure in addition to its management responsibility towards the cluster. At the highest level, the Data Centre Controller (DC) manages the controllers one level below it.

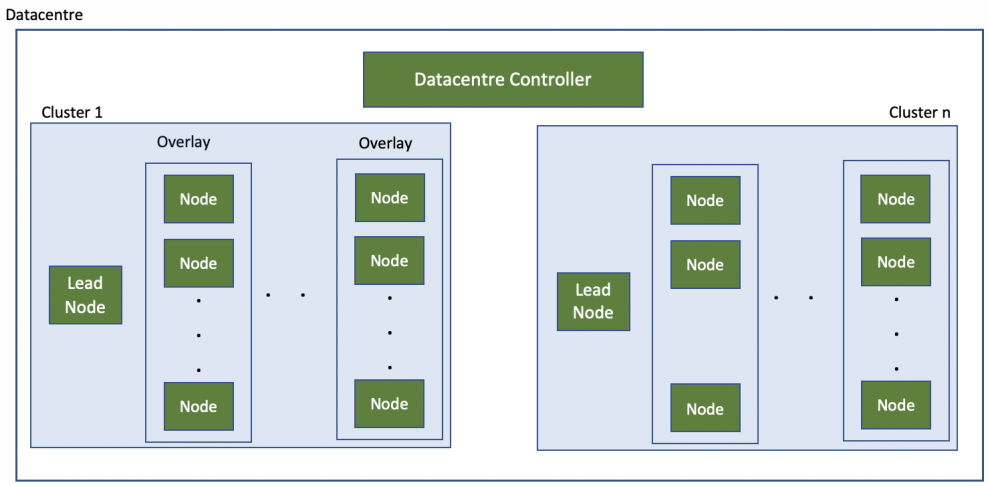



**FIGURE 1** SHDF Escalation Architecture

SHDF attempts to service resource requests at the lowest local level possible, in order to reduce the overhead of servicing the request[37] and to reduce the performance impact of migrating VMs across cluster boundaries[38].

The following subsections describe the components of SHDF and the functionality it provides to the MA.

### 3.2.1 | Overlay Management

Execution nodes operate in a decentralized way within the boundary of a cluster and are organised in multiple overlays. An overlay is a fundamental concept to reliable multicast, as it abstracts the details of the underlying physical network by building a virtual network on top of it, which can be seen as a graph that represents the links between nodes. To construct overlays, typically two main approaches exist[39]: structured and unstructured protocols. Structured approaches tend to be efficient in terms of number of links. However, they are sensitive to node failure, because the overlay structure takes into account metrics such as latency or bandwidth, and upon node failure the structure needs to be rebuilt. In unstructured approaches, links are established randomly among the nodes. To ensure overlay stability, multiple links are established. However, this can cause nodes to receive multiple copies of a given message through its different neighbours.

SHDF is closer to structured approaches[40], where the overlay is constructed to span a subset of a cluster, and by definition is layered on a physical structure of nodes. This solves the proximity challenge in structured approaches, without needing to associate latency metrics with each of the overlay links. When a new node is added to the infrastructure, it gets added to a cluster and assigned an overlay. SHDF enables the MA to determine where a new node is added. When a node is added to an overlay, it receives details of neighbouring nodes from the same overlay, through the data dissemination mechanism described below.

### 3.2.2 | Data dissemination

In centralized and hierarchical architectures, a central component receives state updates from all the nodes under its management to enable the MA to adapt the infrastructure. SHDF removes this dependency and enables the nodes to exchange their state within an overlay. Each node holds a local cache of the state of other nodes in the overlay, and to populate this cache, nodes exchange state messages. A node selects another node at random from the overlay and exchanges its view of the overlay. We used a pull & push gossip approach, where node $x$ sends new state updates to node $y$, and retrieves new states updates at node $y$. A new node state takes O(log N) rounds to reach all nodes, where N is the number of nodes in the overlay[41]. A gossip round completes when every node has initiated an exchange exactly once. The local cache within each node allows the MA to perform VM migration to other nodes within the overlay, and allows the LN to perform VM consolidation across the cluster.

Similar to[42], a NC regularly exchanges state with other NCs and holds a cache of free capacity of other cooperating controllers in the same overlay, shown in Table 1. The gossiping algorithm is shown in Algorithm 1. We enhance this further and make nodes gossip, when there is a state change such as inward migration or a node is being shut down. Each node will keep the latest metrics seen, and every time an exchange of state occurs the timestamp is used to decide if each exchanged entry is later than

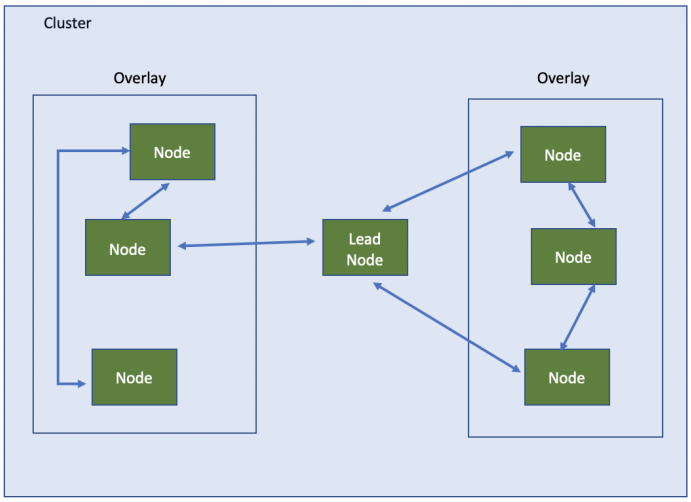



**FIGURE 2** A gossip cycle, within a cluster

**TABLE 1** Exchanged data between overlay nodes

| Field | Description |
|---|---|
| NodeID | The ID of the node, data belongs to |
| VMSize[NumberAvailable] | number of predefined VM sizes that can be hosted on this node |
| TimeStamp | Time stamp for captured metrics |

any existing entries. If it is, the node's internal state is updated, otherwise the data exchanged is discarded. The MA running on each node can use these metrics to enable nodes to cooperate in resource allocation requests, which will be described later.

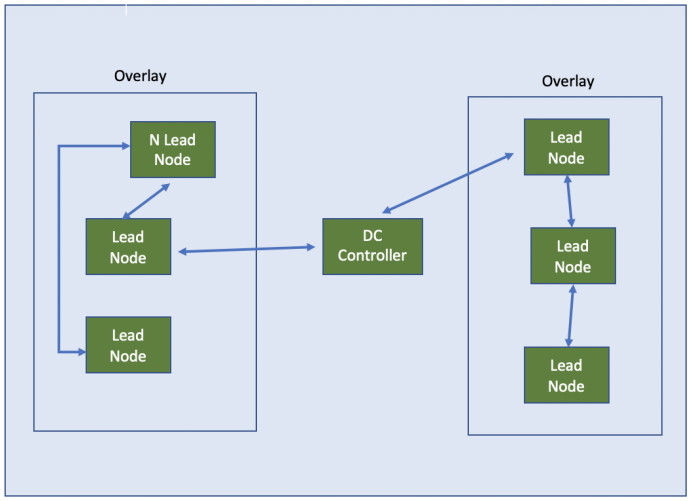



**FIGURE 3** Lead Node Overlays

As in Figure 2, the Lead Node is an ordinary node, and will exchange its own state. The LN is a member of all the overlays in the cluster and will receive state exchanges from all the nodes within the cluster. To restrict the view of a node and reduce

**Algorithm 1** stateExchange@Node

```
1:  procedure EXCHANGESTATE(localState)
2:      overlay ← getOverlayMembers - thisNode
3:      randomMember ← random(overlay)
4:      remoteState ← randomMember.gossip(localState)
5:      for i in remoteState do
6:          remote ← remoteState[i].timeStamp
7:          local ← localState[i].timeStamp
8:          if remote > local then
9:              localState[i] ← remoteState[i]
10:         end if
11:     end for
12: end procedure
13: procedure GOSSIP(remoteState)
14:     respond to caller ← getLocalSate
15:     receivedState ← remoteState
16:     for i in receivedState do
17:         recived ← receivedState[i].timeStamp
18:         local ← localState[i].timeStamp
19:         if received > local then
20:             localState[i] ← receivedState[i]
21:         end if
22:     end for
23: end procedure
```

redundancy of state management, when a node gossips with the LN, it will only receive state updates from its overlay and not from other overlays the LN knows about.

The LN sends aggregated metrics for all the nodes within its cluster, to other cooperating controllers. The LN participates in an overlay with other LNs, in a similar way to NCs, and it can exchange aggregated metric data. The parent controller to the LN is always included in each LN overlay, resulting in state exchanges between the LN in an overlay and the DC controller, in a similar way to the exchanges between a LN and a NC, as shown in Figure 3, resulting in an aggregated view of the whole infrastructure at the DC controller level. This exchanged state data is then used during adaptation actions, which we describe in detail in the VM Migration section.

For the gossip protocol, the exchange frequency and fan out values influence propagation speed and reliability. The fan out value determines the number of times a new state is exchanged by a node. For example, when a node receives an update and the fan out value is 2, the node will exchange this newly received state in the next 2 cycles of the gossip protocol. Through experimentation, we chose a fanout value of 2 as a tradeoff of bandwidth consumption with propagation speed.

### 3.2.3 | Controller functionality - VM migration

When a node cannot satisfy demands of the VMs it hosts, it starts an escalation process that aims to resolve the request at the lowest possible level. The MA running on the stressed node and the LN cooperate to resolve the escalated VM migration, by using our provided framework mechanisms.

Table 2 shows the available methods to support migration and escalation of a VM. SHDF uses different methods to migrate within the overlay and escalate outside it. As shown in Figure 4 and Algorithm 2, when a NC needs to perform a migration it attempts to service this through an increasing escalation set of steps. The process starts within the NC's overlay by sending a request to other nodes within the same overlay (step 1), by using the accumulated metrics of other nodes. If a target node is available and accepts the migration request, then the migration process completes for this cycle. If the selected target node does not accept, other nodes within the overlay are attempted until no further options are available within the overlay. If a target is not available within the overlay, the stressed NC escalates the vm migration request to the LN (step 2), and the MA running on

the LN can query the cluster records from all the overlays, which has state data from all nodes in the cluster, to find a suitable node to house an escalated VM. If the LN locates a target within the cluster, the migration request is forwarded (Step 3). If this does not succeed, then other nodes within the cluster are attempted, in similar way to the overlay step. If the LN can not find a suitable target for the migration within the cluster, it will use its knowledge of other available clusters, through participation in the LNs overlays, to forward the migration request to another LN (Step 4). This target LN will repeat the process performed by the forwarding LN, and attempt nodes within this cluster (step 5). If a suitable target is found the process competes. If the LN in step 4 can not find a target, the request is rejected back to the forwarding LN, which will attempt other LNs in its overlay. If a suitable target is found through another LN , the process completes. If this fails to find a suitable target, the request is escalated to the DC Controllers (step 6). The DC has a view of the entire infrastructure and can forward the request to other LNs (step 7), which the original LN does not cooperate with. This recipient LN will repeat the process carried out by other LN in the escalation chain (step 8), and if a target is found the process completes. If a target is not found, the recent LN will reject the request back to the DC controller (step 9). The DC controller will attempt other LNs, which have not been already tired, until a target is found or all LNs have been attempted. If a target is not found then the infrastructure is highly stressed and the request is rejected back to the original escalating NC. In each of these escalation phases, the MA uses data from the data dissemination to decide on the list of targets to forward a request to.

As requests progress through the escalation process, they are assigned an increasing priority, which can be used by the MA in the decision making process. For example the MA may choose to prioritise finding a host for an escalated VM compared to a new VM placement

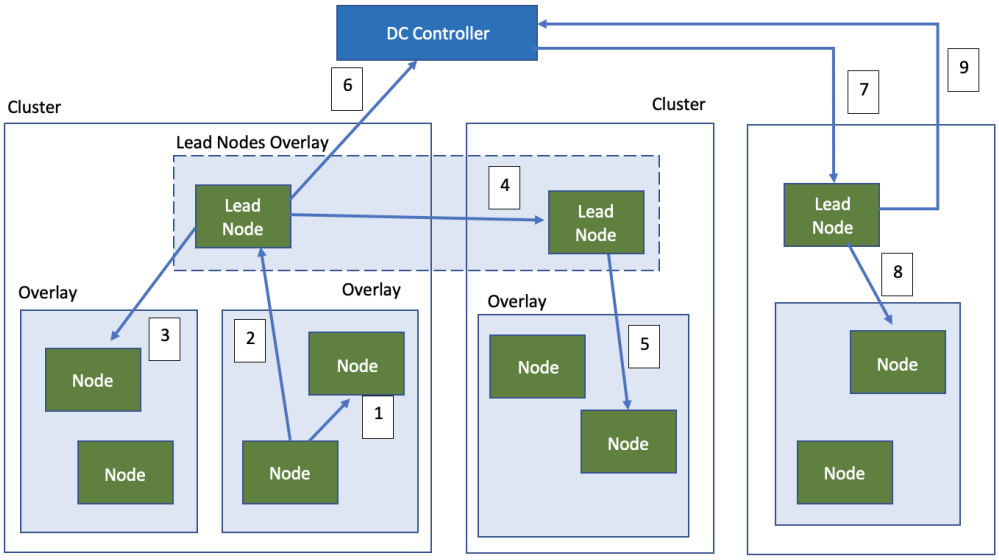



**FIGURE 4** VM migration & escalation process

### 3.2.4 | Controller functionality - Consolidation

At configurable time intervals, 1 hour by default, each of the management controllers, and LNs, can invoke a consolidation process where the MA can examine the state of the infrastructure and for every node under its management, decide to :

- Migrate some VMs from a node
- Migrate all VMs off a node and switch the node off
- No change

The advantage of SHDF is it allows the nodes to primarily operate in a decentralized manner for time sensitive operations such as VM migrations, which could improve SLA violation metrics. For non time sensitive operations like VM placement and

**Algorithm 2** migrateVM@Node

```
1:  procedure MIGRATEVM(localState)
2:      isStressed ← isNodeStressed(localState)
3:      if isStressed then
4:          targets ← validOverlayTargets
5:          vmToMigrate ← MA.orderVMs
6:          targetNode ← MA.findTargetNode(targets, vm)
7:          if targetNode == valid then
8:              invalidateTarget
9:              migrate(vm, targetNode)
10:         else
11:             esclateToLead(vm)
12:         end if
13:     end if
14: end procedure
15: procedure ESCLATETOLEAD(vm)
16:     targetNodes ← validClusterMembers - sendingOverlay
17:     targetNode ← MA.findTargetNode(targetNodes, vm)
18:     if targetNode ≠ null then
19:         invalidateTarget
20:         migrate(vm, targetNode)
21:     else
22:         potentialClusters ← ClustersInOverlay - sendingCluster
23:         targetCluster ← findTargetCluster(potentialClusters, vm)
24:         if targetCluster ≠ null then
25:             targetCluster.esclateToLead(vm)
26:         else
27:             esclateToDataCentreController (vm)
28:         end if
29:     end if
30: end procedure
31: procedure ESCLATETODATACENTRECONTROLLER(vm)
32:     potentialClusters ← allClusters - sendingCluster
33:     targetCluster ← getAvailableCluster(potentialClusters)
34:     targetCluster.esclateToLead(vm)
35: end procedure
```

consolidation, SHDF enables the running of time complex algorithms across parts of the infrastructure or even the whole infrastructure. Compared to typical decentralized approaches[29,30], this will enable the MA to have a larger view of the infrastructure during these operations, leading to opportunities for more optimal VM migration.

## 4 | EXPERIMENT SETUP AND EVALUATION

In our experiments, we evaluate the Quality of Service (QOS) metrics of our proposed hybrid architecture approach versus a centralized, hierarchical and decentralized architectures, with varying workloads and VM configurations. There is an abundance of MAs in the literature that have been implemented using a centralized architecture, and we discuss these in more detail in subsequent sections. We use the same MAs in our comparison of the different approaches, and in doing so we focus the evaluation on the architecture.

**TABLE 2** Controller functionality for Migration of VMs

| Method | Source | Destination |
|---|---|---|
| Migrate VM | Node | Node |
| Escalate VM | Node | LN |
| Place Escalated VM | LN | Node |
| Escalate VM | LN | LN |
| Escalate VM | LN | DC Controller |
| Place Escalate VM | DC controller | LN |

We used simulation to facilitate rapid development of experiments of large data centres. We selected DCSim[43] because of its extensibility and existing implementation of a centralized architecture, allowing us to create baseline comparison for our proposed MF. We additionally implemented 8 MAs, detailed later in this section.

We instantiate SHDF with 3 levels of controllers, running on the root of the data centre (DC Controller), the cluster manager (LN) and leaf nodes (NC).

## 4.1 | Simulator Setup

DCSim allows a VM to use more CPU than reserved, up to an amount that does not impact other VMs. Like[25], we uses a CPU utilisation thresholds of 90% for high, indicating stress level, and we used 60% for low, indicating low utilisation.

In DCSim, an application is modelled as an interactive multi-tiered web application. Each application has a specified client think time, a workload component and a request service time, which is the amount of time required to process each incoming request. The workload defines the current number of clients connected to the application, which can change at discrete points in the simulation based on a *trace* file. The resource requirements are defined as its resource size, which is the expected amount of CPU, memory, bandwidth and storage. DCSim treats bandwidth and storage as fixed requirements, however, CPU requirements can be varied across the simulation based on the VM demands. DCSim applies a cost to VM migration including the time taken for migration, as a function of memory consumed by a VM, and factors in the bandwidth required for the VM migration on the hosting node. Additionally, the boot time of a switched off node has an elapsed time cost. The time taken to switch on a node for migration is reflected on the time period the VM is in a stressed state, and therefore the SLA achieved by a VM. Due to the complexities of building accurate power models, we focus our investigation on scalability metrics.

## 4.2 | Workload, SLA Violations & Energy model

We run the experiments at a load that requires more than 70% of the CPU resources of active nodes. Each experiment is run to simulate 24 hours elapsed time and each simulated application contains a workload trace based on the number of incoming requests to web servers from publicly available traces, we used the following traces included with DCSim Google 1, Google 3, EPA and Clarknet. Figure 5 shows the normalized shape of the workload requests fro each of these traces. We create VMs with different cores and RAM configurations, as shown in Table 3. DCSim is able to track total energy consumption within the data centre, by mapping CPU utilisation of a node to a defined energy consumption amount, and tracking this accumulatively for all nodes.

## 4.3 | Compared MAs

To evaluate our proposed architecture, we build on the work in Mann[44] and implemented similar MAs within DCSim.

Common dimensions that determine how these MAs operate are 1) *Overload detection mechanism*, which is how the MA decides a node is at utilisation point that could impact a metric such as SLA; 2) *VM Selection*, which is the mechanism used to select the VM to be migrated; 3) *Node Selection*, which is the approach used to select the target node for MV migration. Table

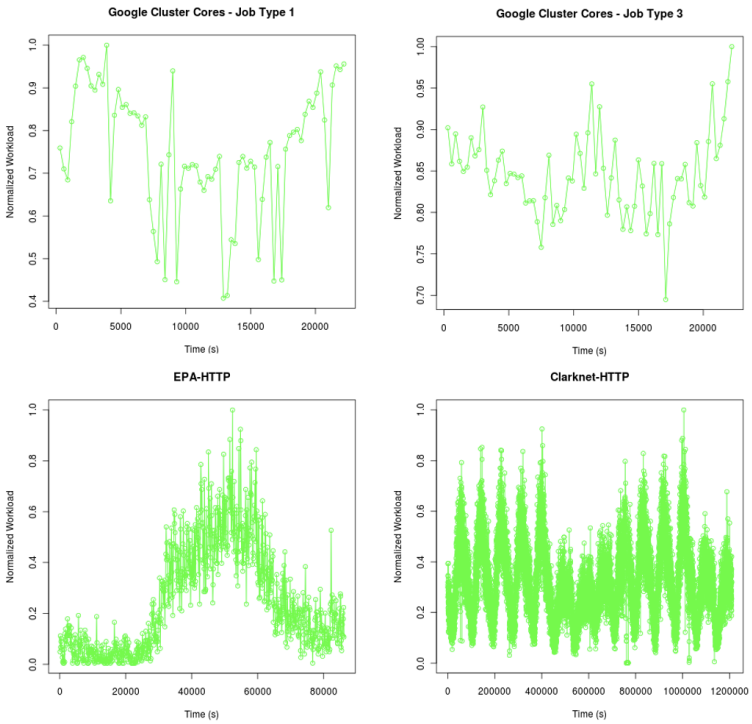



**FIGURE 5** Traces used

**TABLE 3** Experiment configuration

| Config | Config options | Base config |
|---|---|---|
| VM Core (Mhz) | 1000,1300,2500 | Round Robin [1000,1300,2500] |
| Node Capacity (Mhz) | 3000 | 3000 |
| Number of Cores | 1,2 | Round Robin [1,2] |
| VM Memory (MBs) | 1024,2048 | Round Robin [1024,2048] |
| Workload | clarknet, EPA, Google 1, Google 3 | Round Robin [clarknet, EPA, Google 1, Google 3 |
| Number of Nodes | 1000, 2000,3000, 4000, 5000 | 1000 |
| Node stress check frequency | 2 minutes | 2 minutes |
| Application service time | 0.2 seconds | 0.2 seconds |

4, shows these dimensions for each of the compared. The following section briefly describes the management algorithms used in our experiments.

P1[45] uses a set of rules for selecting the next node for a VM migration. P2[46] used a best-fit-decreasing heuristic. P3[47] selects a target node based on historic workload data, and attempts to achieve the smallest risk of demand dissatisfaction. Shi[48] evaluated multiple algorithms, and similar to[44], we used the AbsoluteCapacity (P4) and PercentageUtil (P5). P6[49] created a Modified Worst Fit Decreasing VM Placement. P7[50] uses a Modified Best Fit Decreasing approach to migrate a VM. P8[43] selects VMs that reduce the node stress as candidates for migration, and order candidate target nodes based on partially utilised, under utilised and empty nodes.

**TABLE 4** Dimensions of policies

| MA Identifier | Management Algorithm | Dimension | | |
|---|---|---|---|---|
| | | *Stress Detection* | *VM selection* | *Node Selection* |
| P1 | Lago[45] | Local Regression | Not specified | Highest energy efficiency (plus further rules for tie-breaking) |
| P2 | Guazzone[46] | | Highest CPU load | Highest free CPU capacity (plus further rules) |
| P3 | Demand Risk[47] | | VMs of the most loaded PM | Lowest demand satisifaction |
| P4 | Shi Absloute[48] | | Highest CPU load on the PM with lowest absolute capacity | Highest absolute capacity |
| P5 | Shi Percentage[48] | | Highest CPU load on the PM with lowest utilisation ratio | Highest utilisation ratio |
| P6 | Chowdhury[49] | | Highest CPU load | Highest increase in energy consumption |
| P7 | Beloglazov[50] | | Highest CPU load | Smallest increase in energy consumption |
| P8 | Tighe[43] | | VMs that bring load to below stress | From partially utilised, under utilised, empty nodes |

## 4.4 | Compared MFs

We evaluate our proposed hybrid hierarchical decentralized architecture against a centralized, hierarchical and a decentralized architectures.

The centralized approach[43] has a single central decision making component for VM migration. This controller receives state information from all of the nodes periodically, and on regular intervals invokes an adaption process that checks for stressed nodes and attempts to migrate VMs from stressed nodes to other available nodes within the datacentre.

The hierarchical[25] approach distributes the decision making to multiple controllers. A central data centre controller, a cluster controller and rack controller. The rack controller receives state information from all of the nodes it houses and on regular intervals invokes an adaption process, which checks for stressed nodes and attempts to migrate VMs from stressed nodes. The migration process has some similarity to our escalation process where the rack controller will first to find a target nodes within the rack. If this fail, the rack controller will forward the migration request to the cluster controller. If this fails, the Cluster controller will forward the migration request to the Data Centre controller, which can forward it to other clusters in the datacentre.

The approach in[33] decentralizes the decision making to individual nodes, which cooperate with other nodes to perform VM migration. Each node does not hold state information about other nodes in the infrastructure, and when a node becomes stressed, it broadcasts a migration request to the whole infrastructure and waits for replies. At the end of the wait period, the nodes processes the offers it received and chooses to accept one or power on a new node.

## 4.5 | Modelling the impact of a centralized decision making

DCSim[43] applies a migration cost once a VM is selected for migration, by adding additional time to complete the migration based on the amount of memory used by the VM. However, DCSim does not account for the time it takes to execute the decision making process, or the impact of such time. The length of the decision making process impacts stressed nodes by increasing the amount of time the node stays in a stressed state. In a centralized architecture, all nodes are used as input into the decision making process. Therefore, the execution of decision making could get progressively higher, as the number of managed nodes increases.

To capture the cost of the decision making, we extend DCSim to measure the amount of time during decision making, and add this time to the VM migration duration. As the decision making execution time varies based on the MA and the hardware it is running on, we add a configurable scaling factor that can be applied to the measured execution time.

## 4.6 | Data centre

Our experiments use nodes modelled as ProLiant DL380 G5 Quad Core[51], with 2 dual-core 3GHz CPUs and 16GB of memory. The number of nodes used is specified in each of the experiments, with a minimum of 1,000 nodes. We assume that the data centre supports live VM migration, as this technique is currently supported by most major hypervisor technologies, such VMware[52] or Xen[53].

To minimise the number of variables in our experiments, we chose to keep a homogeneous infrastructure. The various parameters used in our evaluation are outlined in Table 3.

## 4.7 | Evaluation

Our goal is to demonstrate improved QOS metrics compared to centralized, hierarchical and decentralized architectures, and we evaluate these by examining the following metrics: *SLA violations, Energy consumption* and *Total Migration Traffic*.

Simulated applications are modelled as an interactive web servers, running inside a VM. *SLA Violations* are the instances when the allocated CPU is less than then requested CPU. *Total Migration traffic* is the amount of additional data that passes through the network due to VM migration.

We evaluate all architectures under varying scenarios to understand the impact on our chosen metrics. Initially, we evaluate a *mixed workload* scenario to represent the varying workloads deployed on data centres. To understand the impact of specific workloads, we evaluate these individually. We additionally examine scalability and how the approaches cope with dynamic arrival rate of new workload.

### 4.7.1 | No Adaptation

In this experiment, we used a combined workload of all traces in a Round-robin approach, where each created VM uses a different workload trace. We used the centralized architecture with 1000 homogenous nodes and no adaptation is invoked. When a node becomes stressed, it remains stressed for the reminder of the experiment, and Table 5 shows the results.

**TABLE 5** No Adaptation metrics

| Number of SLA Violations | Energy Consmption (KWh) |
|---|---|
| 27053 | 1745.51 |

### 4.7.2 | Mixed workload Assessment

In this experiment, we used a combined workload of all traces in a Round-robin approach, where each created VM uses a different workload trace, in a deterministic order. The mixed workload simulates different workloads that could be experienced in a datacentre. Workload arrival rate is the frequency that new application placement requests arrive at the datacentre. For this experiment we use an arrival rate of 90 new applications per hour, with each application running for 10 hours before shutting down. The experiment simulates 24 hours of elapsed time. We use 1000 nodes and the base configuration in Table 3. We captured metrics for SLA violation, energy consumption and the Total migration traffic.

The SLA violation results using the 8 MAs are shown on Figure 6, and to aid readability we display MAs (P1 to P4) on Figure 6a and MAs P5 to P8 on Figure 6b .

The hybrid architecture achieved lower SLA violation using all MAs, compared to other architectures, with to 29%, 26% and 321% against the decentralized, hierarchical and centralized approaches respectively for the P3 MA. Examining the performance of the MAs, P3 and P7 performed best overall across the different approaches. A contributing factor to the lower numbers of SLA violations in the hybrid architecture, is each target node autonomously accepts incoming VM migration requests based on the current state of the node. In contrast, the centralized and hierarchical architectures use the last seen state collected at an earlier point in time. This could result in target nodes accepting VMs when in a stressed state. When this occurs, it will trigger an additional subsequent migration, which increases the length of time the VM is not receiving the required CPU demand, leading

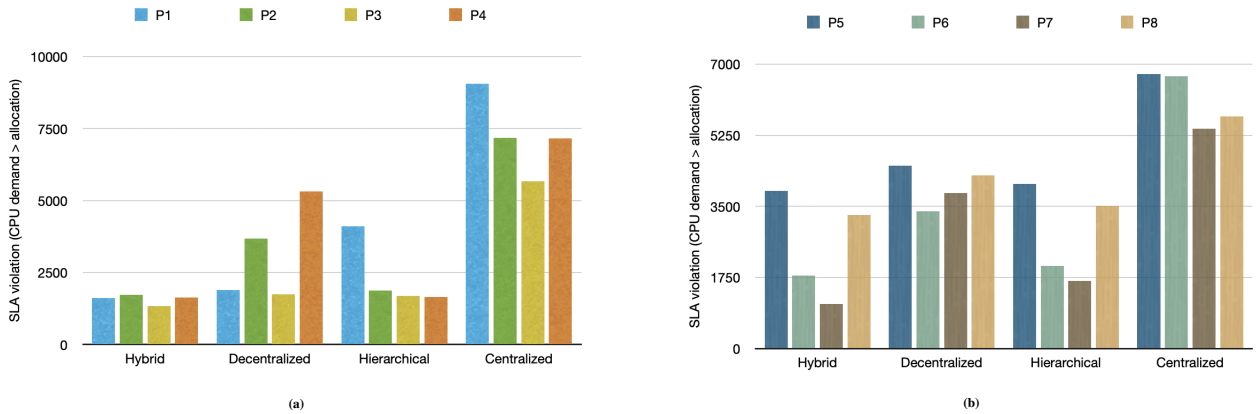



**FIGURE 6** SLA violation instances when demand > allocation, a) P1 to P4 b) SLA Violations P5 to P8

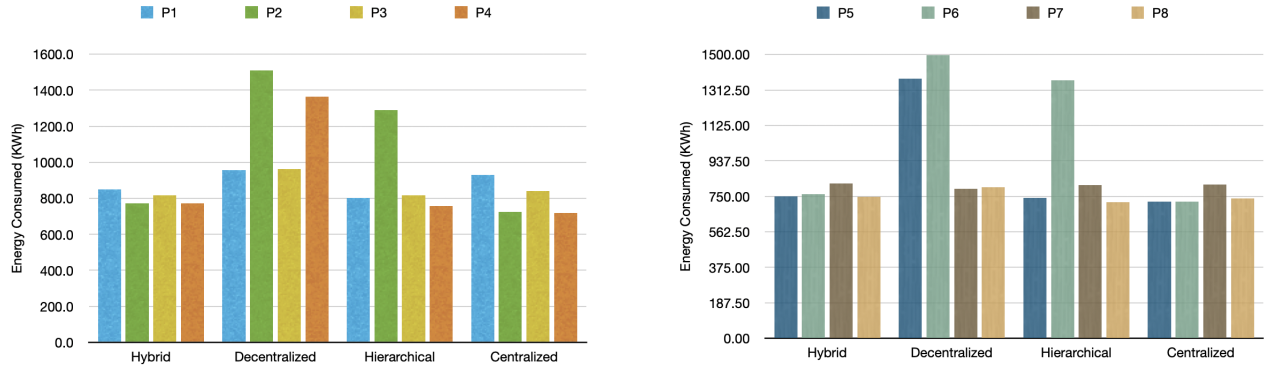



**FIGURE 7** Energy consumption a) P1 to P4 b) P5 to P8

to higher SLA violations. As the centralized architecture uses a single decision making point, it takes longer to make decisions, and it is unsurprising it has the highest number of SLA violations compared to all the other approaches. The decentralized approach relies on broadcast cooperation for VM migration, and thus has a slower decision making process compared to the hybrid architecture, which keeps a state of cooperating nodes within an overlay.

Compared to the *No adaptation* experiment, all architectures reduced SLA violations. Additionally, adaption conserved energy even though it can switch on nodes to perform migration, as the consolidation process can switch nodes off and conserve energy. Adaptation achieved an improvement in energy consumption of up to 128.8%, 16.7%, 28.1%, and 141.9% for the hybrid, decentralized, hierarchical and centralized approaches.

The decentralized approach does not keep local state, and needs to wait on broadcast replies to make a migration and consolidation decision. Additionally the decentralized approach keeps nodes switched on, Figure 7, before they are consolidated and switched off, and thus consumes more energy compared to all other approaches. An advantage of this additional energy consumption is the migrations will not wait for nodes to boot up, compared to the other approaches. This aids the decentralized approach in achieving its SLA violation performance. The Hybrid architecture had a strong performance with a mean difference of 47% and 16.1% lower energy consumption compared to the decentralized and hierarchical approaches respectively. The Hybrid approach had a similar energy consumption profile to the centralized approach.

Examining the migration traffic, shown in Figure 8, the decentralized approach performs lower number of migrations as it has to wait to receive offers for advertised migrations and thus consumes less migration traffic. In contrast, the hierarchical approach performs more migrations to archive its SLA violation performance, and carries out more migrations than Hybrid and decentralized approaches.

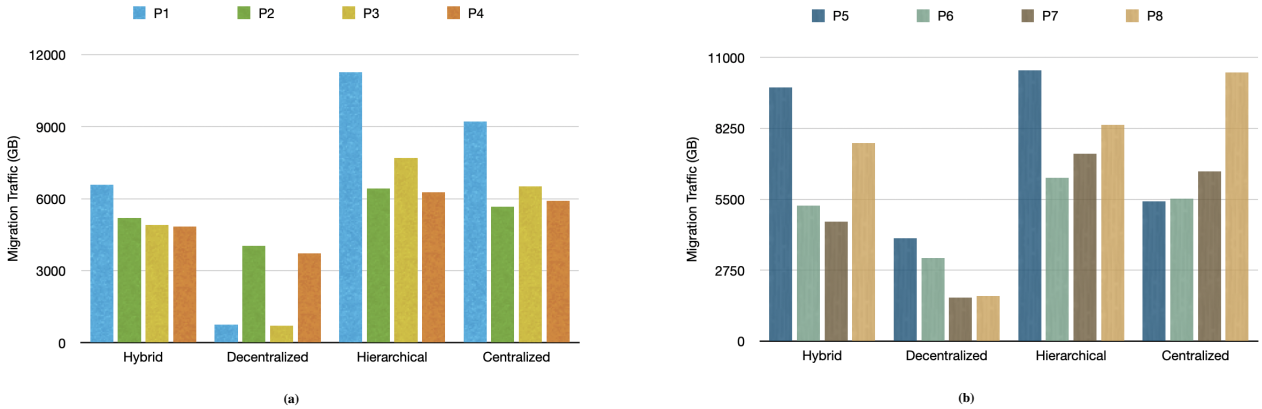



**FIGURE 8** Migration Traffic (GB) a) P1 to P4 b) P5 to P8

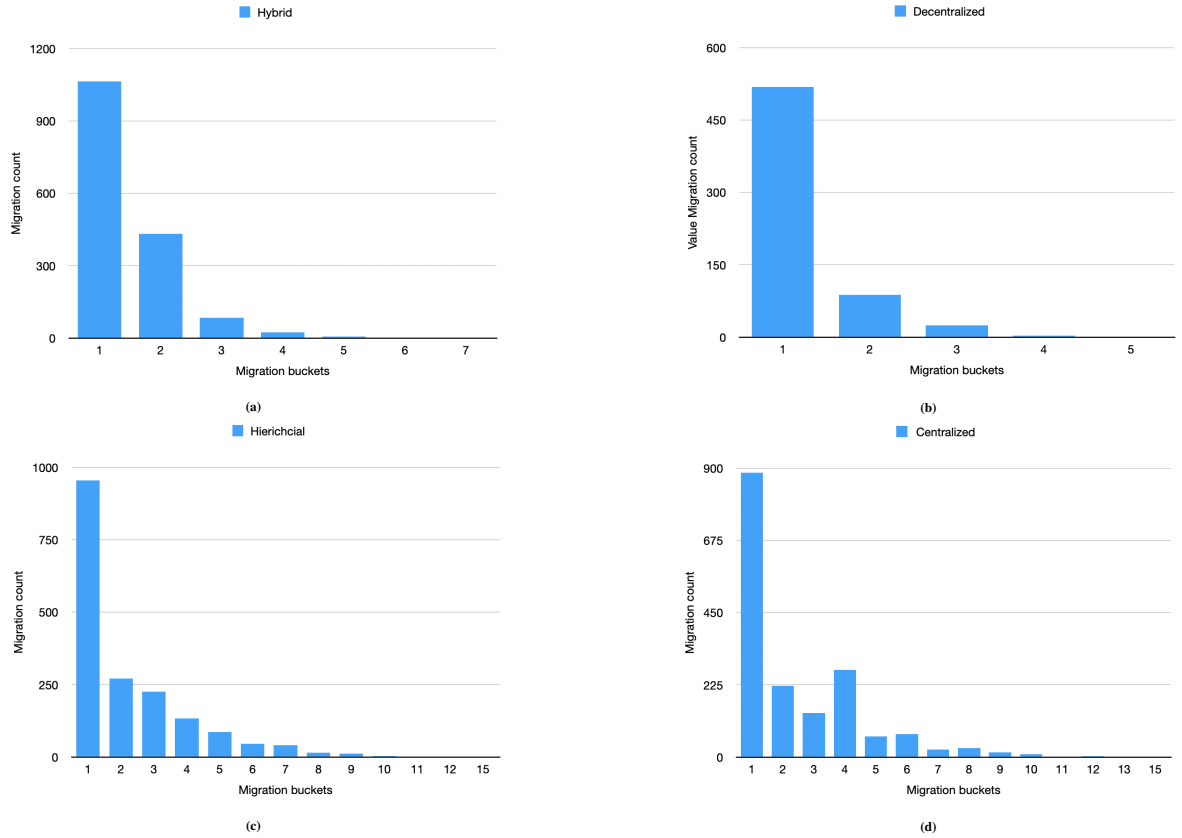



**FIGURE 9** Migration buckets for MA P3 a) Hybrid b) decentralized c) Hierarchical d) Centralized

The hybrid architecture by design uses an escalating level of migrations, starting within the overlay, then within cluster and then to other clusters. The hybrid architecture averaged 97.8% of migrations within the same overlay and the hierarchical averaged 85.5% migrations within the same rack. Therefore, the hybrid approach reduces the migration distance, which reduces the impact of VM migration[54]. In contrast, the centralized and decentralized approaches had no awareness of target node proximity and could choose a target node further away from the source node.

To examine the impact of migration instability[?40,55], a VM being migrated several times during its lifetime, we choose a high performing MA, P3. Figure 9 shows VM migration buckets, representing VMs being migrated exactly once, twice, etc, and the migration count in each bucket.

The P3 MA under the hybrid architecture had 23 occurrences of a VM being migrated exactly 4 times, compared to 272 times in the centralized architecture and 133 in the hierarchical. The decentralized performed less migrations due to the dependency on broadcasting mechanism. The majority of VMs are migrated once in all architectures, however the centralized and hierarchical architectures experience higher VM migration instability, with more VMs being migrated twice or more (cumulative sum of bucket 2 and above), and in the centralized and hierarchical architectures the worst affected VM was migrated 15 times, compared to 7 times in the hybrid architecture.

Another factor influencing higher SLAs in the centralized architecture is the execution time of the decision making process. When a node becomes stressed in the hybrid architecture, the node searches for a target migration node in a smaller search space, starting within the overlay, compared to the global search space in the centralized architecture, and is able to compute a target quicker. As the number of nodes increases, the difference in search space becomes more pronounced. The longer it takes to find a target node, the longer a VM is in a stressed state and thus incurs SLA violations. The hybrid architecture will typically search a fixed number of nodes in the overlay and achieve lower SLA violations.

In summary, the hybrid architecture was able to achieve improved metrics compared to the other approaches, with lower SLA violations, comparable or lower energy consumption and lower total migration traffic.

### 4.7.3 | Workload impact

To investigate the impact of workload further, we use the individual workloads from the previous experiment and evaluate them individually. For this experiment we use an arrival rate of 90 new applications per hour, with each application running for 10 hours before shutting down. The experiment simulated 24 hours of elapsed time. We use the base configuration in Table 3.

The results for the Google 1 workload are shown in Figure 10, and show the effect on SLA violations and energy consumption. We chose the best performing MAs from the previous experiment, P3 and P7. This workload has an average of 0.74 normalized load and thus high stress. The hybrid approach outperformed other approaches by 118%, 28.5% and 428.7% for the decentralized, hierarchical and centralized respectively (P3 MA). The hybrid approach performs even better on this workload, compared to the mixed workload, indicating that more stressful workloads benefit form the speed of the decision making process in the hybrid approach.

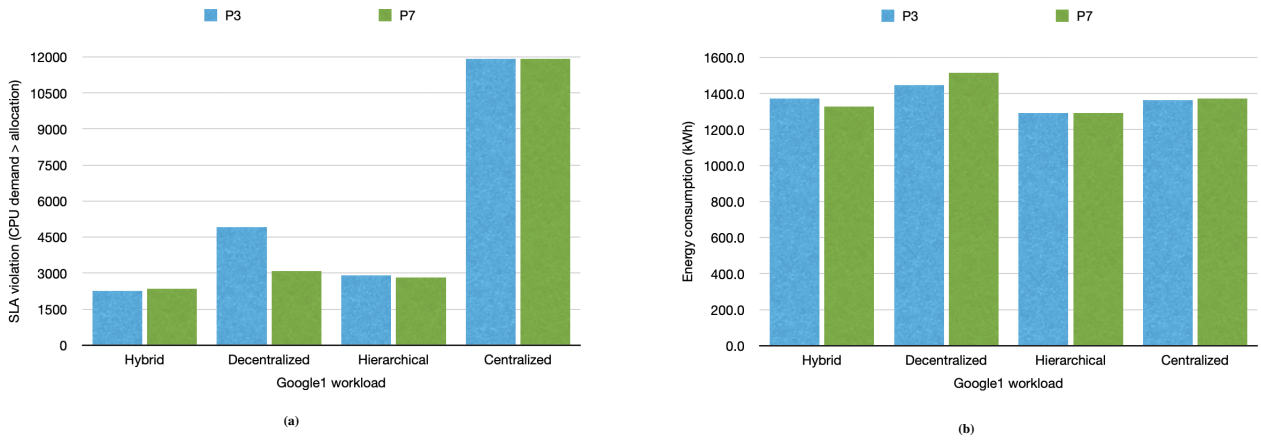



**FIGURE 10** Google 1 workload evaluation a) SLA violations b) Energy consumption

Examining the energy consumption, the hybrid approach consumes 6% and 1% more energy compared to the hierarchical and centralized approaches for the P3 MA, shown in figure 10b. This is due to the faster decision making process in the hybrid and is offset by the significantly better SLA violation performance, Figure 10a.

The results for the Google 3 workload are shown in Figure 11, and show the effect on SLA violations and energy consumption. This workload has an average of .83 normalized load and is the highest workload stress we have evaluated. The hybrid approach outperformed other approaches by 434%, 92.% and 796% for the decentralized, hierarchical and centralized respectively (P3 MA). The hybrid approach performs even higher on this workload, compared to the mixed workload, indicating that more stressful workloads benefit form the speed of the decision making process in the hybrid approach.

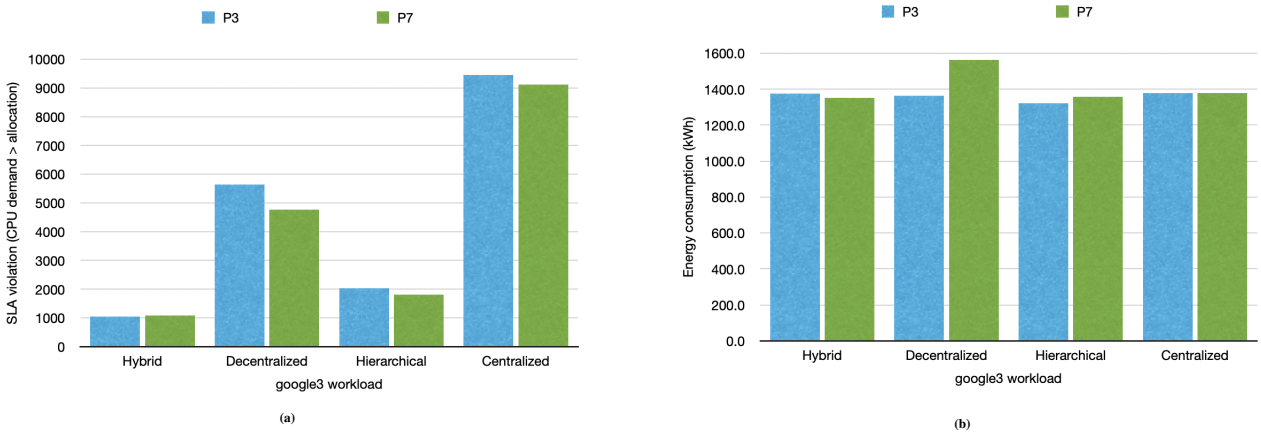



**FIGURE 11** Google 3 workload evaluation a) SLA violations b) Energy consumption

The Clarknet workload has an average of 0.31 normalized workload, and is lower average stress compared to the Google 1 and Google 3 workloads. The results for SLA violations and Energy consumption are shown in Figure 12. The hybrid approach achieves lower SLA violations compared to the hierarchical and centralized approaches, but not against the decentralized approach, which has nodes switched on by default to satisfy the workload demands. This results in the decentralized consuming 72%, 67% and 79% more energy compared to the hybrid, centralized and hierarchical approaches respectively, for the P7 MA. However for the P3 MA, the decentralized approach was able to achieve the lowest SLA violations and least energy consumption for this workload.

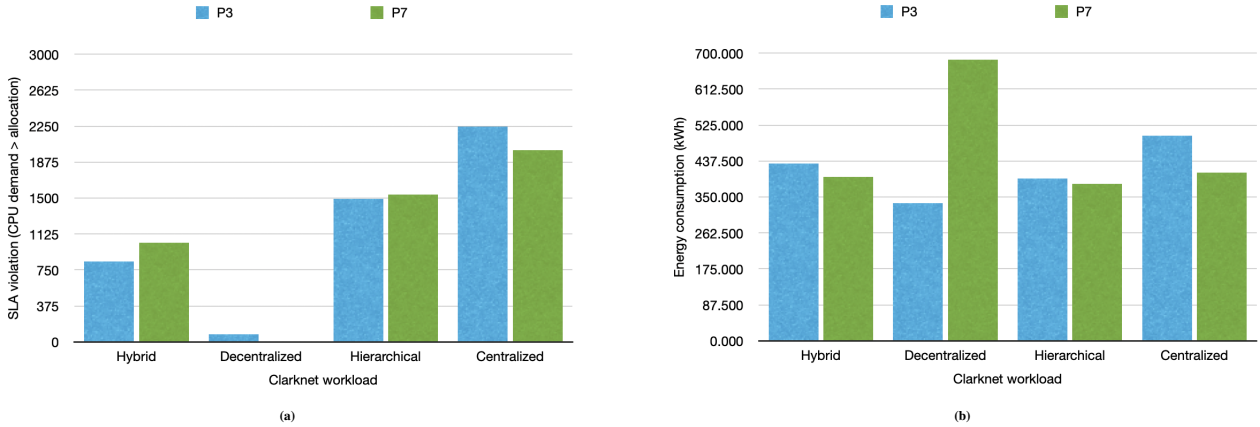



**FIGURE 12** Carknet workload: (a) SLA violations b) Energy consumption

The EPA workload has an average of 0.24 normalized workload, and is lower average stress compared to the Google 1 and Google 3 workloads. The results for SLA violations and Energy consumption are shown in Figure 13. Similar to the Clarknet workload, the hybrid approach achieves lower SLA violations compared to the hierarchical and centralized approaches, but not against the decentralized approach, which has nodes switched on by default to satisfy the workload demands. In contrast the decentralized P3 MA has consumed 209%, 219% and 331% more energy compared to the hybrid, centralized and hierarchical approaches respectively. A similar effect occurred on the decentralized P7 MA, which achieved the lowest SLA violations out of all the approaches, but used 163%, 200% and 189% more energy compared to the hybrid, hierarchical and centralized respectively.

Examining the results of the individual workloads, we conclude that on stressful workloads (Google 1/Google 3) hierarchical based approaches (hybrid/hierarchical) are able to provide rapid decision making, which leads to improved SLA violation performance compared to the decentralized and centralized approaches. In contrast, on low stress workloads (EPA/Clarknet) the

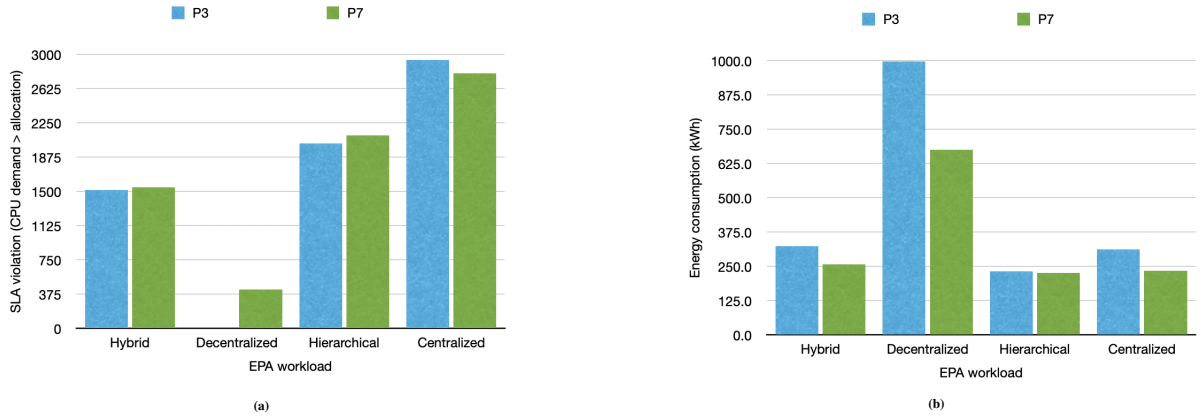



**FIGURE 13** EPA workload: (a) SLA violations b) Energy consumption

decentralized approach with more nodes switched on is able to achieve lower SLA violation, trading this with significant energy consumption. The hybrid approach is able to achieve a balanced SLA violation performance and energy consumption tradeoff.

### 4.7.4 | Dynamic arrival

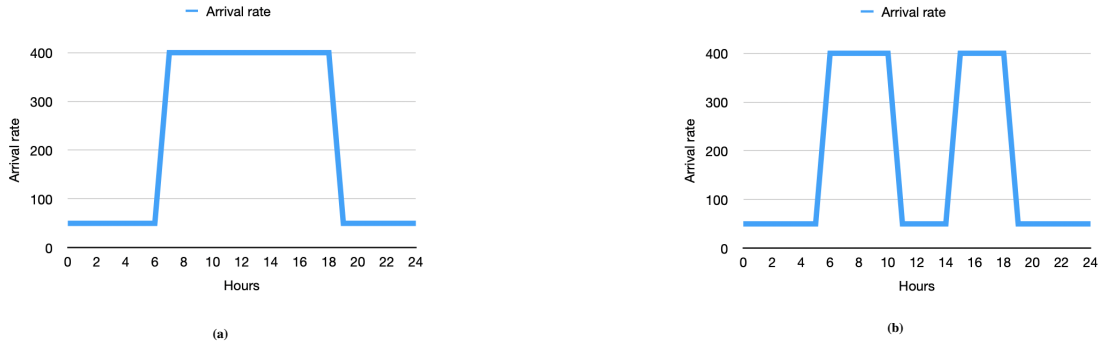



**FIGURE 14** Arrival rate patterns

Up to this point we evaluated the approaches using a steady arrival rate for new workload. To understand the impact of rapid and sharp arrival rate, we examine two scenarios, Figure 14. In the first scenario (a), there is a small and steady flow of new VM creation requests, followed by a single sharp increase in number of VM creation requests that persists for several hours, followed by a return to the previous small steady number of VM creation requests. The second scenario (b), is similar to the first scenario but with 2 sharp increases in the arrival rate of new VM creations. Each application runs for 10 hours before shutting down. We use the P3 MA and google 3 trace. The experiment simulates 24 hours of elapsed time, and we use the base configuration in Table 3.

The high arrival rate of VMs on the infrastructure, as each VM runs the Google 3 workload, increases the VMs that begin to experience a stress state where they do not deliver the requested CPU demand, and thus enter SLA violation. The results are shown in Figure 15, and show the distributed approaches (decentralized, hybrid and hierarchical) outperform the centralized approach. The hybrid approach has a rapid decision making process and was able to achieve the lowest SLA violations compared to all approaches. The hierarchical approach and decentralized approach had comparable performance. The disadvantage of the broadcasting mechanism was countered by nodes being switched on in the decentralized. On the two spike scenario, the decentralized lagged in the first spike, but was able to switch additional nodes to outperform the hierarchical approach on the second spike. It achieved this at the cost of energy consumption, utilising 42%, 55% and 72% more energy compared to the hybrid, hierarchical and centralized respectively.

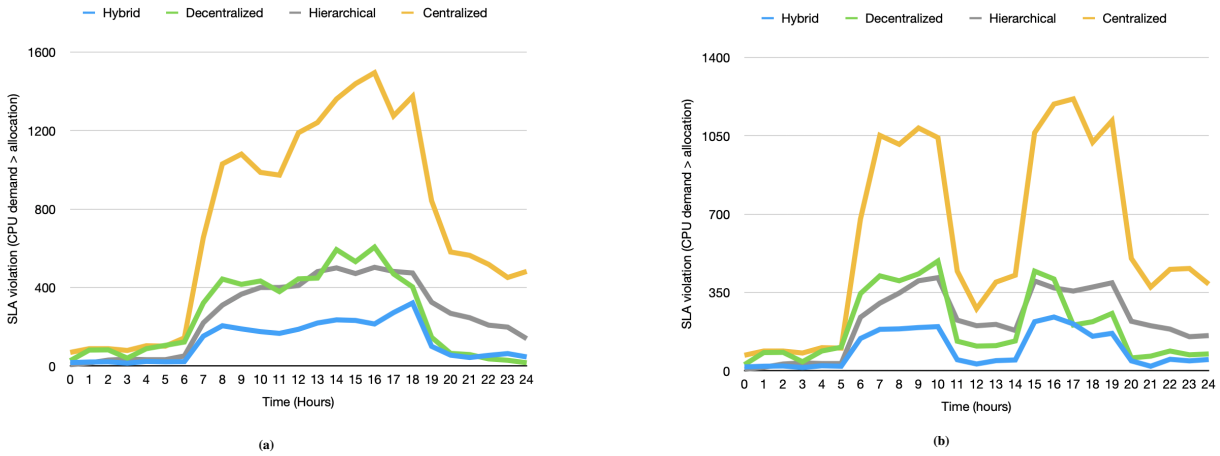



**FIGURE 15** High spike in arrival rate: a) 1 spike b) 2 spikes

### 4.7.5 | Scalability

To evaluate how the approaches scale, we use the most stressful workload, Google 3, and maintain the stress ratio for each node, by increasing the load and the number of nodes in the datacentre. Similar to other experiments, we simulate 24 hours, and we use the P3 MA; the results are shown in Table 6.

**TABLE 6** SLA violations with increasing number of nodes

| ***Number of Nodes*** | **Hybrid** | | **Decentralized** | | **Hierarchical** | | **Centralized** | |
|---|---|---|---|---|---|---|---|---|
| | *SLA* | *Energy* | *SLA* | *Energy* | *SLA* | *Energy* | *SLA* | *Energy* |
| 1000 | 1054 | 1376.1 | 5638 | 1362.1 | 2129 | 1322.2 | 9447 | 1379.0 |
| 2000 | 2085 | 2724.7 | 6871 | 4100.8 | 4707 | 2643.8 | 16975 | 2521.8 |
| 3000 | 2948 | 4099.0 | 10179 | 5410.1 | 7605 | 3956.6 | 25741 | 3917.6 |
| 4000 | 4144 | 5427.3 | 12384 | 8302.1 | 10636 | 5313.7 | 37261 | 5570.6 |
| 5000 | 6089 | 6624.4 | 12829 | 10989.5 | 14163 | 6656.2 | 51666 | 6958.8 |

The hybrid approach maintains its SLA performance as more nodes and VMs are added to the data centre, with lower SLA violations than all other approaches. The hierarchical approach initially outperforms the decentralized in SLA violations, but the latter catches up at 5000 nodes. The decentralized trades its SLA violation performance for energy consumption, using 65.1% more energy compared to the hierarchical approach (5000 nodes). As expected, the distributed approaches outperform the centralized approach.

## 5 | CONCLUSIONS

In this paper, we have presented a scalable architecture for managing cloud resources, that combines the benefits of both the hierarchical and decentralized approaches in a hybrid manner. Our escalation approach attempts to service resource requests at the lowest local level possible, in order to reduce the overhead of servicing the request. We evaluated and demonstrated that our proposed architecture achieves significantly improved QOS metrics compared to centralized, hierarchical and decentralized architectures, particularly in high stress workloads

In the hybrid architecture, nodes are autonomous and decide when to accept migration requests, and are typically less stressed compared to nodes in the evaluated architectures. This results in lower VM migration instability, and enables more opportunities for VM consolidations. We have shown these factors lead to lower SLA violations and less migration traffic, when combined

with a variety of MAs from the literature, and the hybrid architecture retains the benefits of the hierarchical and decentralized approaches.

In the future, we plan to continue investigating resource allocation decision making, by implementing and incorporating a MA that can take further advantage of the specific feature features of the hybrid architecture.

## References


1. Hameed A, Khoshkbarforoushha A, Ranjan R, et al. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing* 2016; 98(7): 751–774. doi: 10.1007/s00607-014-0407-8

2. Herbst NR, Kounev S, Reussner R. Elasticity in Cloud Computing: What It Is, and What It Is Not. In: 10th International Conference on Autonomic Computing. ; 2013: 23-27.

3. Jiao L, Tulino AM, Llorca J, Jin Y, Sala A. Smoothed Online Resource Allocation in Multi-Tier Distributed Cloud Networks. *IEEE/ACM Transactions on Networking* 2017; 25(4): 2556-2570. doi: 10.1109/TNET.2017.2707142

4. Ruprecht A, Jones D, Shiraev D, et al. VM Live Migration At Scale. *SIGPLAN Not.* 2018; 53(3): 45–56. doi: 10.1145/3296975.3186415

5. Hummaida AR, Paton NW, Sakellariou R. Adaptation in cloud resource configuration: a survey. *Journal of Cloud Computing* 2016; 5(1): 1–16.

6. Chowdhury MR, Mahmud MR, Rahman RM. Implementation and performance analysis of various VM placement strategies in CloudSim. *Journal of Cloud Computing* 2015; 4(1): 20. doi: 10.1186/s13677-015-0045-5

7. Zhang F, Tang X, Li X, Khan SU, Li Z. Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems* 2019; 98: 672 - 681. doi: https://doi.org/10.1016/j.future.2018.09.009

8. Tan B, Ma H, Mei Y. Novel Genetic Algorithm with Dual Chromosome Representation for Resource Allocation in Container-Based Clouds. In: IEEE 12th International Conference on Cloud Computing (CLOUD). ; 2019: 452-456

9. Hu Y, Zhou H, Laat dC, Zhao Z. Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. *Future Generation Computer Systems* 2020; 102: 562 - 573. doi: https://doi.org/10.1016/j.future.2019.08.025

10. Zhang Y. *Virtualization and Cloud Computing*ch. 2: 13-36; John Wiley & Sons, Ltd . 2018

11. Hummaida AR, Paton NW, Sakellariou R. SHDF - A Scalable Hierarchical Distributed Framework for Data Centre Management. In: 16th International Symposium on Parallel and Distributed Computing (ISPDC). ; 2017: 102-111

12. Islam S, Lee K, Fekete A, Liu A. How a Consumer Can Measure Elasticity for Cloud Platforms. In: ICPE '12. Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering. Association for Computing Machinery; 2012; New York, NY, USA: 85–96

13. Lehrig S, Eikerling H, Becker S. Scalability, Elasticity, and Efficiency in Cloud Computing: A Systematic Literature Review of Definitions and Metrics. In: QoSA '15. Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. Association for Computing Machinery; 2015; New York, NY, USA: 83–92

14. Ghahramani MH, Zhou M, Hon CT. Toward cloud computing QoS architecture: analysis of cloud systems and cloud services. *IEEE/CAA Journal of Automatica Sinica* 2017; 4(1): 6-18.

15. Joshi N, Shah S. A Comprehensive Survey of Services Provided by Prevalent Cloud Computing Environments. In: Satapathy SC, Bhateja V, Das S., eds. *Smart Intelligent Computing and Applications*Springer Singapore; 2019; Singapore: 413–424.

16. Megahed A, Nazeem A, Yin P, Tata S, Nezhad HRM, Nakamura T. Optimizing cloud solutioning design. *Future Generation Computer Systems* 2019; 91: 86 - 95. doi: doi.org/10.1016/j.future.2018.08.005

17. Addis B, Ardagna D, Panicucci B, Squillante MS, Zhang L. A Hierarchical Approach for the Resource Management of Very Large Cloud Platforms. *IEEE Transactions on Dependable and Secure Computing* 2013; 10: 253-272.

18. Jung G, Hiltunen MA, Joshi KR, Schlichting RD, Pu C. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In: International Conference on Distributed Computing Systems. IEEE; 2010; Washington, DC, USA: 62-73.

19. Zhu X, Young D, Watson BJ, et al. 1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center. In: International Conference on Autonomic Computing. IEEE; 2008; Washington, DC, USA: 172-181.

20. Almeida J, Almeida V, Ardagna D, Cunha Í, Francalanci C, Trubian M. Joint admission control and resource allocation in virtualized servers. *Journal of Parallel and Distributed Computing* 2010; 70: 344-362.

21. Moens H, Famaey J, Latre S, Dhoedt B, Turck FD. Design and Evaluation of a Hierarchical Application Placement Algorithm in Large Scale Clouds. In: IFIP/IEEE International Symposium on Integrated Network Management. ; 2011: 137-144.

22. Leontiou N, Dechouniotis D, Denazis S, Papavassiliou S. A hierarchical control framework of load balancing and resource allocation of cloud computing services. *Computers and Electrical Engineering* 2018; 67: 235 - 251. doi: https://doi.org/10.1016/j.compeleceng.2018.03.035

23. Goudarzi H, Pedram M. Hierarchical SLA-Driven Resource Management for Peak Power-Aware and Energy-Efficient Operation of a Cloud Datacenter. *IEEE Transactions on Cloud Computing* 2016; 4(2): 222 - 236.

24. Moens H, Turck FD. A Scalable Approach for Structuring Large-Scale Hierarchical Cloud Management Systems. In: 9th International Conference on Network and Service Management (CNSM). ; 2013: 1-8.

25. Keller G, Tighe M, Lutfiyya H, Bauer M. A hierarchical, topology-aware approach to dynamic data centre management. In: Network Operations and Management Symposium (NOMS). ; 2014: 1 -7.

26. Van HN, Tran FD, Menaud JM. SLA-aware virtual resource management for cloud infrastructures. In: . 02. IEEE International Conference on Computer and Information Technology. IEEE; 2009; Washington, DC, USA: 357-362.

27. Mola O, Bauer M. Towards Cloud Management by Autonomic Manager Collaboration. *International Journal of Communications, Network and System Sciences* 2011; 4(12A): 790-802.

28. Hindman B, Konwinski A, Zaharia M, et al. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In: NSDI'11. Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation. USENIX Association; 2011; USA: 295–308.

29. Wuhib F, Stadler R, Spreitzer M. Dynamic resource allocation with management objectives: implementation for an OpenStack cloud. *IEEE Transactions on Network and Service Management* 2012; 9(2): 213-225.

30. Sedaghat M, Hernández-Rodriguez F, Elmroth E, Girdzijauskas S. Divide the Task, Multiply the Outcome: Cooperative VM Consolidation. In: IEEE International Conference on Cloud Computing Technology and Science. IEEE; 2014; Washington, DC, USA: 300-305.

31. Calcavecchia NM, Caprarescu BA, Di Nitto E, Dubois DJ, Petcu D. DEPAS: a decentralized probabilistic algorithm for auto-scaling. *Computing* 2012; 94(8): 701–730.

32. Pantazoglou M, Tzortzakis G, Delis A. Decentralized and Energy-Efficient Workload Management in Enterprise Clouds. *IEEE Transactions on Cloud Computing* 2016; 4(2): 196-209.

33. Tighe M, Keller G, Bauer M, Lutfiyya . A distributed approach to dynamic VM management. In: Proceedings of the 9th International Conference on Network and Service Management. ; 2013: 166 to 170.

34. Loreti D, Ciampolini A. A Decentralized Approach for Virtual Infrastructure Management in Cloud. *International Journal on Advances in Intelligent Systems* 2014; 7(3/4): 507-518.

35. Nouri SMR, Li H, Venugopal S, Guo W, He M, Tian W. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Generation Computer Systems* 2019; 94: 765 - 780. doi: doi.org/10.1016/j.future.2018.11.049

36. Barrett E, Howley E, Duggan J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience* 2013; 25(12): 1656-1674. doi: doi.org/10.1002/cpe.2864

37. Maurer M, Brandic I, Sakellariou R. Adaptive resource configuration for Cloud infrastructure management. *Future Generation Computer Systems* 2013; 29(2): 472-487.

38. Aldhalaan A, Menascé DA. Autonomic Allocation of Communicating Virtual Machines in Hierarchical Cloud Data Centers. In: Cloud and Autonomic Computing (ICCAC), 2014 International Conference on. ; 2014: 161-171.

39. Matos M, Sousa A, Pereira J, Oliveira R, Deliot E, Murray P. *CLON: Overlay Networks and Gossip Protocols for Cloud Environments*: 549–566; Berlin, Heidelberg: Springer Berlin Heidelberg . 2009.

40. Ganesh AJ, Kermarrec AM, Massoulié L. HiScamp: Self-organizing Hierarchical Membership Protocol. In: EW 10. Proceedings of the 10th Workshop on ACM SIGOPS European Workshop. ACM; 2002; New York, NY, USA: 133–139.

41. Birman K. The Promise, and Limitations, of Gossip Protocols. *SIGOPS Oper. Syst. Rev.* 2007; 41(5): 8–13. doi: 10.1145/1317379.1317382

42. Wuhib F, Yanggratoke R, Stadler R. Allocating Compute and Network Resources Under Management Objectives in Large-Scale Clouds. *Journal of Network and Systems Management* 2015; 23(1): 111–136.

43. Tighe M, Keller G, Bauer M, Lutfiyya H. DCSim: A Data centre simulation tool for evaluating dynamic virtualized resource management. In: Network and service management (CNSM), 2012 8th international conference and 2012 workshop on systems virtualization management (SVM). ; 2012: 385–392.

44. Mann ZÁ, Szabó M. Which is the best algorithm for virtual machine placement optimization?. *Concurrency and Computation: Practice and Experience* 2017; 29(10): e4083–n/a. e4083 cpe.4083doi: 10.1002/cpe.4083

45. Lago DGd, Madeira ERM, Bittencourt LF. Power-Aware Virtual Machine Scheduling on Clouds Using Active Cooling Control and DVFS. In: MGC '11. Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science. Association for Computing Machinery; 2011; New York, NY, USA

46. Guazzone M, Anglano C, Canonico M. Exploiting VM migration for the automated power and performance management of green cloud computing systems. In: 1st International Workshop on Energy Ecient Data Centers. ; 2012: 81–92.

47. Calcavecchia NM, Biran O, Hadad E, Moatti Y. VM Placement Strategies for Cloud Scenarios. In: IEEE Fifth International Conference on Cloud Computing. ; 2012: 852-859

48. Lei Shi , Furlong J, Runxin Wang . Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In: 2013 IEEE Symposium on Computers and Communications (ISCC). ; 2013: 000009-000015

49. Chowdhury MR, Mahmud MR, Rahman RM. Study and performance analysis of various VM placement strategies. In: IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). ; 2015: 1-6

50. Beloglazov A, Buyya R. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. *Concurrency and Computation: Practice and Experience* 2012; 24: 1397-1420.

51. HPE ProLiant. 2016.

52. VMware. http://www.vmware.com; 2016.

53. Citrix . Xen. http://www.xenserver.org; 2016.

54. Zhang W, Han S, He H, Chen H. Network-aware virtual machine migration in an overcommitted cloud. *Future Generation Computer Systems* 2017; 76: 428 - 442. doi: https://doi.org/10.1016/j.future.2016.03.009

55. Tso FP, Hamilton G, Oikonomou K, Pezaros DP. Implementing Scalable, Network-Aware Virtual Machine Migration for Cloud Data Centers. In: IEEE Sixth International Conference on Cloud Computing. ; 2013: 557-564

## 4.4 Scalable Virtual Machine Migration using Reinforcement Learning

Abdul R Hummaida, Norman W Paton and Rizos Sakellariou

**Publishing state:** Published. Journal of Grid Computing, 2021.
DOI: https://doi.org/10.1007/s10723-022-09603-4

**Summary**: Given the dynamic properties of cloud environments and the regular change in the structure of workloads and access patterns, we hypothesise that a dynamic MA that can regularly learn how to choose a target node to migrate a stressed VM, will be well suited to balancing the goals of meeting SLAs and conserving energy consumption. Our goal is to propose a MA that can be combined with the proposed hybrid MF, which has been shown to scale and manage a large infrastructure. In this paper, we investigate and propose a reinforcement learning MA, which can integrate well into our hybrid MF and achieve fast convergence and lower SLA violations compared to heuristics.

We model applications as a single VM that runs a CPU intensive activity such as web a server in a PaaS context, where we can capture response time. When migration is applied it is always to the single VM that houses the application. However, the proposed MA can be extended to manage applications made up of multiple VMs by collectively instantiating the VMs that make up an application. In the simulation, VM response time is recorded periodically, by calculating the length of time it takes for requests to complete. Factors that influence the request completion time include the required VM resources, the assigned resources and the load generated by the used trace file.

The RL decision making is engaged periodically, and by default every 2 minutes, each node will autonomously assess its state and use the RL method to decide on an action. As part of this process, if a VM is not meeting its response time target, RL will migrate the VM away from a node. There are multiple VM selection approaches in the literature and in this section we choose a simple approach that selects the most stressed VM to migrate. In Section 4.5 we explore another VM selection approach that aims to optimize the migration point.

The hybrid MF includes a set of escalation steps, where the migration process considers the overlay, cluster and wider infrastructure respectively. In this paper, we

extended the original hybrid architecture, in which the escalation was only performed when no target nodes were available in the overlay. In this paper, individual nodes can escalate at any point using a reinforcement learning (RL) action. This enables the RL agent to explore escalation before the overlay becomes full, and reduce utilization pressure in the overlay by leaving capacity and thus reducing SLA violations.

Our chosen RL method introduces a small computational and memory overhead to decision making. The computational processing includes maintaining the Q-table for each of the states, and the lookup and update of the table. To reduce the memory overhead we have used a state reduction approach that results in the Q-table becoming a matrix of 2 states and 13 actions, $2 \times 13$, resulting in a minimal memory overhead. The RL method takes into account the cost of the additional energy consumption in the target node. However as the RL method has a fixed processing cost, we abstract this from the cost calculation.

To examine the impact of migration instability, a VM being migrated several times during its lifetime, we evaluated the stability of the proposed MF (Section 4.3). This shows the hybrid MF performs fewer migrations of the same VM compared to other architectures, which is achieved through autonomous decision making in each node. While the Hybrid MF can reduce VM migration instability, it does not have explicit instability detection and mitigation. This can be achieved by extending the decision making to include cost benefit analysis and prediction of the future state of a potential migration target.

Each experiment is run to simulate 24 hours of elapsed time and each simulated application contains a workload trace from the public traces included in DCSim. The simulation is designed to instantiate a certain number of VMs per hour, where each VM will run one of the public traces in a round-robin approach. For example, when 1000 VMs are created per hour and 5 traces are used, 200 VMs will use 1 of the traces. Each trace sets the size of incoming traffic to a VM and typically changes every 5 minutes. As the round-robin approach is used in the experiments for each MA and MF evaluation, we assume this is a valid method to instantiate the VMs. Each VM runs for 10 hours before shutting down.

**Key contributions:** Contribution 4 (see Section 1.4).

**Grid Computing manuscript No.**
(will be inserted by the editor)

# Scalable Virtual Machine Migration using Reinforcement Learning

***Abdul Rahman Hummaida · Norman W Paton · Rizos Sakellariou**

Received: date / Accepted: date

**Abstract** Solutions based on Reinforcement Learning (RL) have been presented to manage cloud infrastructure, however, these tend to be centralized and suffer in their ability to maintain Quality of Service (QoS) for data centres with thousands of nodes. To address this, we propose a reinforcement learning management policy, which is able to run decentralized, and achieve fast convergence towards efficient resource allocation, resulting in lower SLA violations compared to centralized architectures. To address some of the common challenges in applying RL to cloud resource management, such as slow learning and state/action management, we use parallel learning and reduction of the state/action space. We have also demonstrate unique, multi-level reinforcement learning cooperation, that further reduces SLA violations. We use simulation to evaluate and demonstrate our proposal in practice, and compare the results obtained with an established heuristic, demonstrating significant improvement to SLA violations and higher scalability.

**Keywords** Reinforcement learning · Data centre Scalability · Virtual Machine migration · Hierarchical architecture · Distributed architecture

## 1 Introduction

Cloud computing is an established paradigm to give end users access to computing resources through a simplified as-a-service model. Cloud Providers (CPs) build data centres and abstract resources through a virtualisation layer, with a Virtual Machine (VM) as a common form. End users request these services through APIs, that map requests to virtual resources that reside on physical resources in the data centre.

VM placement, both to schedule initial virtual machines and to adapt the placement, to meet assorted goals, has been a subject of extensive investigation [17,37,33,45,50]. To perform VM placement, specific resource utilisation or compatibility requirements are typically required, such that a VM is mapped onto a physical node within the data centre. From

*Abdul Rahman Hummaida E-mail: abdul.hummaida@postgrad.manchester.ac.uk · Norman W Paton E-mail: norman.paton@manchester.ac.uk · Rizos Sakellariou E-mail: rizos@manchester.ac.uk
University of Manchester, Department of Computer Science, Kilburn Building, Oxford Rd, Manchester M13 9PL, UK

within the pool of physical nodes that satisfy VM constraints, the mapping process becomes an optimisation problem that aims to increase resource utilisation, energy consumption or profit. Post initial VM placement, it is common for the cloud environment to undergo a load change, where the initial VM constraints are no longer met. CPs use adaptation methods to continuously monitor and perform VM placement [23].

VM placement is accepted as an NP-Hard problem and heuristic solutions have been used to solve it [61]. However, data centres are becoming increasingly large, which means the problem of making globally appropriate placement decisions is increasingly challenging. Many of the solutions to manage cloud infrastructure are centralized and suffer in their ability to support data centres with thousands of nodes. Furthermore, heuristic approaches have been shown to have scalability challenges [26,53], in their ability to execute the decision-making process with increasing size of data centre infrastructure. Different approaches have been proposed to address this problem of scale [62,48,11,56,44]. These tend to be decentralized heuristics, with no central controller, and have been shown to manage large number of nodes. A key challenge with heuristics is that their performance depends on multiple factors including the statistical patterns of resource demands, and if the underlying scenario changes, heuristics may start to perform poorly [21].

In this paper, we build on a hybrid architecture that has been shown to have benefits of rapid decision making [24], fewer SLA violations, lower network traffic utilisation and improved scalability as the number of nodes in the data centre increases, compared to centralized, hierarchical and other decentralized architectures. The hybrid architecture has hierarchical decentralized controllers operating at different scopes. On the lowest level, controllers dynamically adjust resource configurations and cooperate with other nodes and higher level controllers in performing VM placement. However, although the hybrid architecture supports localised decision-making in a global context, this still raises the question as to the policies, which map VMs to data centre resources, that should be applied at different levels in the hierarchy. The most suitable policy may depend on subtle features of the infrastructure and the workload, hence there may be benefits from learning the policy. Reinforcement Learning (RL) is an approach that develops or refines a policy in the light of experience [54]. In RL, an agent performs learning by interacting with an environment, and learning through trial and error. The agent takes actions and observes the outcome of these actions. RL has been applied successfully in a variety of resource management settings [15,55,51]. Here we develop an RL approach for the hybrid architecture [24] and apply it to cloud resource management. Multiple RL agents operate in a decentralized way and share the learning from migration's actions. Additionally, the agents uniquely cooperate at multiple management hierarchy levels to achieve rapid decision making and learn an optimal online policy. Our approach is shown to reduce SLA violations and achieve high scalability.

The contributions of this paper are as follows:

1. A realisation of a RL strategy in a specific hybrid architecture, with lower SLA violations and high scalability compared to a Heuristic based approach.
2. The utilisation of the hybrid architecture, to achieve unique, parallelised multi-level RL agent cooperation.
3. An empirical evaluation of the RL strategy in comparison with a heuristic strategy that has been shown to be effective in practice [7]. This shows the proposed RL approach improves SLA violation performance, compared to the heuristic approach, and further improvements are achieved when RL is combined with the hybrid architecture from our earlier work [24].

The rest of this paper is organised as follows. We first describe some of the challenges in designing an efficient resource management controller. Section 3 describes related work in cloud resource management, and Section 4 describes a background into RL. Section 5 summarises the hybrid architecture from our earlier work, and Section 6 describes our proposed RL approach. Section 7 presents an evaluation of our implementation and compares it to a heuristic hybrid, RL centralized and a heuristic centralized approaches. In Section 8 we draw conclusions and discuss future work.

## 2 Problem Statement

Cloud Providers (CPs) provide access to resources that are typically pooled and shared with multiple customers, with a layer of orchestration that separates individual customer usage. Physical resources are abstracted through virtualisation technology into compute, memory, storage and networking with logical separation of these resources, and typically presented as a Virtual Machine (VM). CPs customer workloads experience variability and VMs are created and deployed onto physical nodes to run customer workloads. CPs need to re-optimize the infrastructure regularly to provide high levels of availability and reliability.

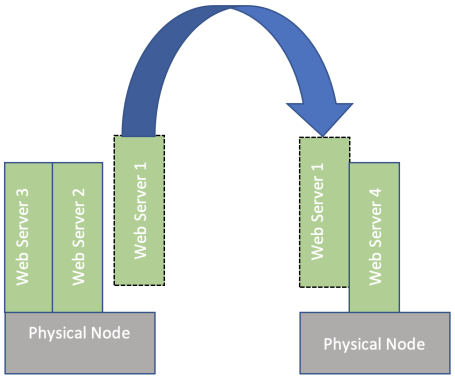



**Fig. 1** Managing SLAs

An illustrative use case is VMs running web applications, such as e-commerce systems, which are typically N-tier and with web servers that process business logic. The resources assigned to the VM and the amount of incoming requests determine the *response time* experienced by end users. The web server will typically experience a variable arrival rate of requests. Using Figure 1 as an example, Web Server 1 can experience an increase in requests, which will increase the resources used by Server 1 and increase the load on the physical node, and in turn could start to impact the response time for Web Server 2 and Web Server 3. Continuing with this state could have an adverse affect on all of the web servers. CPs aim to react to such conditions in order to preserve the customer's SLAs. A CP's goal is to enable such VMs to operate at a rate that meets customer agreed SLAs, and balance this with the CP's operating costs. We research and solve the following problems:

- Detect when a VM is stressed.
- Identify which VMs to migrate.

- Apply a decision making approach to optimize the migration of a VM, and choose a target node to host the VM in such a way that brings response time within SLA levels.
- Develop an architecture for the control system that monitors and optimises the migration of VMs.

In particular, we aim to solve these problems in ways that scale to work on data centres with thousands of nodes. Our goal is to design a scheme that minimises the SLA violations, by detecting stressed VMs and actioning a migration plan to choose new target nodes to house the VMs. One option is to move stressed VMs to a newly switched on node or a lightly loaded node, however this could have a negative impact on the amount of energy consumption in the data centre and thus impact CP's operating costs. Therefore, the optimisation scheme needs to balance meeting SLAs with energy consumption.

Data centre workloads vary, with new workloads being provisioned and other workloads being deprovisioned, so an efficient controller needs to cope with these varying demands. Additionally, web servers typically exhibit varying traffic loads that can cause nodes to become stressed and lead to SLA violations. When this occurs, the controller needs to engage a migration plan to reduce SLA violations.

In a large data centre managing thousands of VMs and nodes, a control scheme needs to be scalable and continue to be responsive. Decentralized architectures that have autonomous monitoring, management and feedback are well suited to large scale environments [58]. In this paper, we have developed a control approach where each node operates autonomously and contains a learning agent that shares learnt knowledge with other agents.

## 3 Related Work

This section outlines related work on VM placement and on the use of machine learning, and in particular reinforcement learning (RL) for resource management.

VM Placement is mapping of customer workloads to infrastructure resources, in a way that achieves a particular objective, such as reducing energy consumption or load balancing, while ensuring SLAs are met [34]. VM placement consists of two parts: *initial* VM placement, which refers to the first allocation of VMs to nodes in the data centre, and VM *migration* or relocation, which involves the revision of an earlier placement decision. VM placement performs the mapping in order to meet an SLA, energy or profit goal and has been studied extensively. Meeting an SLA objective can be seen as detection of an overloaded node, selection of VM(s) on the overloaded node, identification of a target node, and engaging the hypervisor to migrate the VM to the identified target node in order to avoid performance degradation. When a node is deemed overloaded and the adaptation process chooses to migrate a VM from a stressed node, VM(s) need to be chosen for migration. VM selection approaches include: Minimum Migration Time chooses a VM that takes the shortest time to complete the migration, Random Selection chooses a VM based on a uniformly distributed discrete random variable and Maximum Correlation selects a VM of the highest correlation of CPU utilization with other VMs to migrate [7]. In this paper, we use a similar approach to Maximum Correlation to select a VM for migration, by sorting the VMs and choosing the VM with highest correlation.

Heuristic approaches are widely used in the literature for VM migration. The authors in [22] proposed an approach to increase efficiency of node utilization and balance utilization of CPU and memory usage on active nodes across the data centre. The approach used a multi-dimensional resource usage model for target node selection and used it to guide

the VM placement process. A resource usage factor is assigned to each node and used in node selection. Their experiments show minimisation of low utilized resources and more balanced utilization of CPU and memory usage on active nodes. Uniquely, the authors in [20] considered joint VM and container migration. The approach divides the cloud resource management problem into sub-problems including over-load/under-load detection, identifying if a VM or a container should be migrated, VM/container selection and migration of the VM/container. Local Regression was used to detect overloaded nodes and VM selection was done using Minimum Migration Time [7]. Target nodes for migrations were selected using SLA-aware allocation. The authors in [63] proposed a multi-constraint optimization model by considering migration cost and remaining runtime of VM migration, and used a heuristic policy. The applied constraints were the total CPU/memory requirements of VMs allocated should not exceed the node's resource capacity and a VM should be assigned to a single physical node; maximum duration that a node can be in SLA violation and the remaining runtime for a VM were also considered. However, heuristics typically do not find a globally optimal answer but may provide locally optimal outcomes [67,42].

Cloud environments are highly complex, and are typically multi-tenanted with non linear workloads; as a result they experience high variability. Machine Learning (ML) techniques can offer an opportunity to adjust resource management in a dynamic way, which is reflective of the context of cloud environments [32]. ML techniques can be categorised as *Supervised Learning* where every data sample is labeled and used as input. The learning process works by associating features of the input and human feedback. In *Unsupervised Learning* samples are used as input, but unlike supervised learning, there are no labels and the learning process aims to learn the data distribution within the sample. For example VM usage patterns can be used to cluster VMs into distinct groups through unsupervised learning. In *Reinforcement Learning* there is no labeled input, instead an agent learns dynamically from its environment and balances exploration of new knowledge versus exploitation of known knowledge.

Some ML approaches focus on auto-scaling resources, autonomously provisioning and deprovisioning resources. The authors in [41] presented an auto-scaling method for adaptive provisioning of elastic cloud services, based on ML time-series forecasting and queuing theory, aimed at optimizing response time. The approach uses Support Vector Machines (SVM) to predict the average node load for the following hour, and then use this with a queuing model to adjust the resources assigned to a node. Their experiments show SVM has better prediction than moving average and linear regression. Similarly, another prediction approach was presented in [64] with Long Short Term Memory (LSTM) time-series prediction, and provisioning through queue theory. Their results show LSTM performed better in terms of prediction accuracy than the SVM and Autoregressive Integrated Moving Average. A Neural Network technique was presented in [60], which proposes an adaptive selection that can choose a VM consolidation approach based on the current environment and the cloud provider's priority on energy and SLA violation. The approach firstly generates a raw dataset by simulating the methods for several time steps. Each row will contain the initial environment parameters and normalized evaluation result of all policies. The results (energy and SLA violations) for each row then are normalized. A performance score is calculated using the evaluation priority and normalized evaluation result from the raw dataset; this score is then used to train the neural network. Another framework for resource reservation is presented in [52], based on load prediction and several ML approaches including Neural Networks, Linear Regression, RepTree and M5P. The approach takes an initial reservation plan and monitoring data as inputs and optimises the plan based on monitoring data from observations, with CPU being the main monitored resource. The evaluation showed RepTree

was able to learn faster than the Neural Network; however, the Neural Network ultimately yielded better predictions.

While ML models approaches have shown to be effective, one limitation of these approaches is that they are typically trained offline and require retraining to make use of new data [49]. Cloud environments are dynamic and exhibit regular changes in the structure of workloads and access patterns. Aptly, RL can operate online and learn dynamically from interacting with a changing environment and make use of new information to enhance the decision making process. Additionally, RL approaches do not require prior knowledge of the optimization model and are not coded explicit instructions relating to which action to take next; instead, they learn actions through feedback from the environment. These features make RL well-suited to cloud resource management resource management [41].

Auto-scaling of the assigned VMs is the focus of some of the approaches in the literature, by using RL to add more resources for customer workloads. The authors in [28] propose a general purpose model-free learning algorithm, based on Q-learning, that adapts to unknown system specifics, such as application traffic, to generate scaling up or down actions. Our proposal also uses Q-learning, however we focus on migrating stressed VMs as opposed to auto-scaling. To speed up the convergence of RL, the authors in [6] developed an approach that parallelises Q-learning to speed up convergence of agents in order to achieve auto-scaling of VMs. We propose a similar approach of parallel learning and further enhance it with cooperative learning between agents running at different layers of a hierarchical cloud infrastructure. The problem of autonomous scaling of cloud resources can be mapped to MAPE-K architecture (Monitor, Analyse, Plan, and Execute) [19]. The approach enhanced the performance of the planning phase and a planning module uses linear regression to predict future demands, with Q-learning performing dynamic resource allocation. Their experiments show the approach increases the resource utilization and decreases the total cost while reducing SLA violations. Our proposal has many similarities with monitoring, planning and execution modules, although, unique to our approach, is cooperative learning between RL agents running at different layers of a hierarchical infrastructure.

Reinforcement learning techniques can suffer from the curse of dimensionality, where the state and/or action space grows exponentially, which introduces challenges in the time needed for the RL agent to explore a given environment and introduce space complexity in memory consumption by the agent. To address this, some approaches utilise function approximation, such as Deep Q-Learning (DQN) [40], which is an approach of combining deep learning and Q-learning to combat the challenges of Q-learning in environments with a large or continuous state action space. The authors in [30] propose a DQN based model to respond to anomalies in CPU and memory bottlenecks and apply granular actions to autoscale resources. The approach in [9] also proposes a Deep Q-learning based approach to adjust the size of a cluster, by taking the state of the cluster as input and training an RL agent to resize a cluster based on administrator defined policies and rewards. The agent can use Deep Q-learning, Double Deep Q-learning or Full Deep Q-learning, and the approach was compared to other RL and decision-tree based approaches and shows it gains rewards up to 1.6 times better. Alternatively to using a DQN, the approach in [46] used a coarse-grained Q-table and can achieve higher resolution in the Q-table with less cost. The approach proposed granular actions to adjust CPU and memory resources, and applied it in a distributed learning mechanism using Q-learning. The work in [10] used a heuristic method to reduce the state space to a smaller set, by dividing the original state space into multiple exclusive subsets, where a range of states can fit into the same subset, thus reducing the state space to aid RL convergence speed. Other non-statistical approaches for function approximation have been proposed [25,5,8]. To address the curse of dimensionality, instead of function

approximation we used an aggregation approach in our proposal. This reduces states and actions into smaller groups, with multiple states being mapped into a smaller number of states and actions.

**Table 1** Adaptation proposals. Key: Q: Q-learning, DQN: Deep Q-Learning, ML: Machine Learning, SVM: Support Vector Machines, QT: Queueing Theory, SARSA: State–Action–Reward–State–Action

| MA | Objective | Considered Resource | | | | Techniques |
|---|---|---|---|---|---|---|
| | | ***CPU*** | ***Memory*** | ***Storage*** | ***Network*** | |
| Yadav[65] | SLA & Energy | x | | | | Heuristic |
| Gholipour[20] | SLA & Energy | x | | | | Heuristic |
| Xu [63] | SLA & Energy | x | x | | | Heuristic |
| Gupta [22] | Energy | x | x | | | Heuristic |
| Vozmediano[41] | SLA | x | | | | SVM & QT |
| Barrett [6] | SLA | x | x | x | | Q |
| Ghobaei-Arani [19] | SLA | x | x | x | x | Q |
| Moghaddam [30] | SLA | x | x | | | DQN |
| Rao [46] | SLA | x | x | | x | Q |
| Bitsakos [9] | SLA | x | x | x | | DQN |
| Bibal [8] | SLA | x | x | | | SARSA |
| Jamshidi [25] | SLA & Energy | x | x | | | Q |
| Bu [10] | SLA | x | x | | | Q |
| Arabnejad [5] | SLA | x | | | | Q & SARSA |
| Ren [47] | SLA & Energy | x | x | | | DQN |
| Ren [66] | SLA | x | x | x | | DQN |
| Nouri [43] | SLA | x | x | | | Q |
| Witanto [60] | SLA & Energy | x | | | | Multiple |
| Sniezynski [52] | SLA & Energy | x | | | | Multiple ML |

Migration of stressed VMs, which are failing a quality of service metric, to another target node, is the focus of some of the work in the literature and has a similar aim to our proposal. The authors in [47] propose a Deep RL based framework that performs VM migrations and uses a proximal policy optimization (PPO) algorithm and a neural network based on LSTM for function approximation. The architecture of the approach is split onto an offline and an online part. The offline part trains an agent by sampling log data, which is generated by the online agent. The online agent has a similar method to the offline, except it does not update the agent parameters. Online decision information is used for the next offline training. Our approach also caters for reducing SLA violations, however we choose a different state action reduction approach to manage the challenges with Q-learning. Additionally, the approach was only evaluated with a small number of VMs, while our experiments were evaluated with thousands of VMs and nodes. The authors in [43] propose a Q-learning controller to manage complex workload arrival patterns and use a decentralized architecture, with each node responsible for maintaining its own SLA performance. The approach is able to add nodes and scale down by shutting down excess nodes to save on energy consumption. To combat the state space challenge in Q-learning, the approach uses a reduced state space. Similarly, our proposal uses a decentralized architecture, applies knowledge sharing among the RL agents, uses aggregation to reduce the state action space, and uses linear regression to monitor QoS metrics like response time. However, the uniqueness of our approach is cooperative learning between RL agents running at different layers of a hierarchical cloud infrastructure. The authors in [66] investigate VM migration during data centre upgrade and use a DQN to decide a target node for each VM migration with the objective of min-

imising the total migration time. We use a state action aggregation approach to address the dimensionality challenge, while the authors use function approximation. Table 1 presents a summary of the approaches used in VM cloud resource management, including heuristics and ML techniques.

While there have been attempts at examining the scalability of approaches based on RL [10,46], these tend to be at a small scale that is not representative of the size of the infrastructure in public clouds. We propose a highly scalable RL approach and examine its ability to manage a large infrastructure, with many thousands of nodes.

## 4 RL Background

Reinforcement Learning (RL) is a machine learning technique that enables an agent to learn within an interactive environment, through trial and error, and uses signals from the environment in a feedback loop. In Figure 2, the agent monitors the current state of the environment (Step 1), and chooses an action from the available options on the environment (Step 2). The environment will then generate a reward for the action taken by the agent, and transition to a new state (Step 3). The goal oriented agent aims to learn the set of actions, a policy, that will lead it to a specific goal, or to maximise an objective function. RL problems are typically formulated with well defined transition probabilities and modelled as a Markov decision process (MDP)[54].

While RL has shown much promise, there are significant challenges applying to practical real world problems [16], including limited offline training logs, learning on the real system where exploration has to be limited and delays in the system actuators to gather action reward.

RL approaches are categorised as model based or model free methods, depending on whether full knowledge of the model can be specified. Model based approaches need knowledge of the environment model, while model free methods learn a policy based on observations and rewards [54].

There are two common control categories of RL. *Value-based or off-policy methods*: RL algorithms proceed to learn the value function for every state/state-action pair to arrive at the optimal policy. Q-learning is a common algorithm in this category. *Policy-based or on-line methods* directly learn the parameters for the policy, instead of learning an explicit policy function, by fine tuning a vector of parameters in order to select the best action to take for policy. SARSA is a common example in this category.

Deep reinforcement learning combines RL with Deep Neural Network based approximation of expected values. An offline phase prepares the network with prior system knowledge, for example from execution. These are then used during online RL execution to select best actions based on the state of the environment [35].

Q-learning [59] is a common control strategy in cloud resource management, due to its simple implementation. Q-learning is model free RL algorithm, belonging to a collection of algorithms known as Temporal Difference (TD) methods. Q-learning estimates the optimal action value function, independent of the policy being followed, and does not require a full model of the environment. The action-value function or Q-function is updated using Equation 1, and approximates the value of selecting a certain action at a certain state.

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma MaxQ((s_{t+1}, a_{t_{+1}}) - Q(s_t,\ a_t))] \tag{1}$$

In this equation, $\alpha \in [0,1]$ is the learning rate, or step size, and determines how the agent learns from recent updates. A high value for $\alpha$ means the most recent information obtained

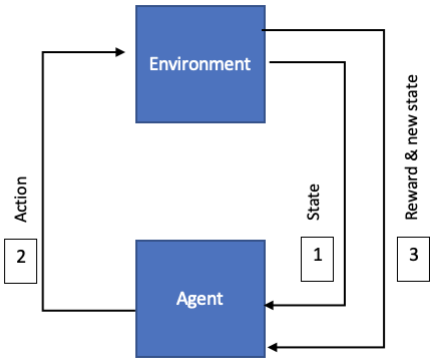



**Fig. 2** RL continuous process

is utilised while a low value implies slower learning. To dampen the reward's effect on the agent's choice of action, the discount factor $\gamma \in [0,1]$ is used. When $\gamma$ is set to 1, the agent will emphasise greater weight to rewards in the future. When it is closer to 0, the agent will only consider the most recent rewards. $MaxQ(s_{t+1}, a_{t_{+1}})$ returns the maximum estimate for the future state-action pair. Once the Q-Value is calculated it is then stored in the agent's Q-Table.

One of the challenges in action selection is exploration vs exploitation, which is the challenge of choosing to further explore the environment for possibly better rewards or choosing to exploit the known reward paths. To speed up the process of learning an effective policy, the agent needs to exploit rewarding actions, but it needs to also find these online. A common approach is $\varepsilon$-greedy [54], which selects the action with the highest estimated reward most of the time. With a small probability of $\varepsilon$, we choose to explore, and not exploit by randomly selecting an available action, independent of the action-value estimates we have previously learned. Other action selection methods include soft-max and optimistic initialisation of values [54].

Some of the challenges with RL include poor initial performance, large training time, and large state space. To improve the poor initial performance, human experts can set initial values for a given state/action [38], and convergence time can be reduced by using parallel learning agents [8], where each agent learns from its experience of visited states, and learns the value of unvisited states from other agents, and a Q-table can be shared among all the agents. A high number of states and or actions would lead to complex Q-tables with millions of cells and consume large amount of memory. Exploring all the states to generate a Q-table can also be time consuming [27]. To solve the challenges with large state and action space, techniques such as tile coding and function approximation can be used [29,36,12]. Other approaches to reducing the problem of dimensionality include aggregation, where multiple states or actions are aggregated to a smaller number of abstracted categories [19,8,10].

In this paper we develop an RL based controller to solve the VM migration problem and combine Q-Learning with an aggregated state action space to address the curse of dimensionality in Q-learning. To speed up RL convergence, we utilise parallel learning agents that learn from a shared collective experience of all agents. We develop a reward function that focuses on learning a policy to reduce SLA violations, and balance this with energy consumption.

## 5 Hybrid Architecture

Cloud Providers (CPs) build and operate large scale data centres that contain numerous computing resources, that are typically virtualised and require a level of orchestration of the shared resources, which is a challenging issue [3]. We classify the management process into two dimensions, *Management Algorithm (MA)* and *Management Framework (MF)*. The MA is responsible for deciding how workloads are assigned to infrastructure resources, while the MF enables the MA to execute by providing common functionality, such as hierarchy level management and aggregation of metrics between nodes. The combined functionality results in workloads executing on infrastructure nodes. In our previous work we detail a hybrid MF [24], which we summarise here. A new RL based MA is the focus of this paper.

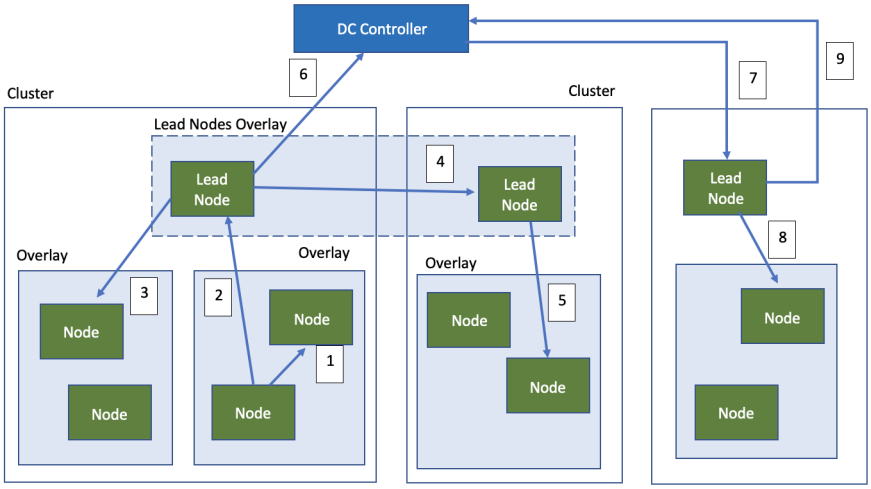



**Fig. 3** VM migration & escalation process

The Management Framework (MF) provides common utilities that enable the MA to execute, including a mechanism to propagate node state, and a decision engine architecture that may be centralized, hierarchical or decentralized. The architecture of our previous work [24] consists of hybrid hierarchical decentralized controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a Node Controller (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. A collection of NCs form a cluster of nodes. Each NC cooperates with a Lead Node (LN), which is a higher level controller for all the NCs within a given cluster. Unique to our proposal, the NCs within each cluster are divided into logical groups, called overlays, where a NC cooperates with other NCs within the same overlay. Each NC exists in only one overlay and in one cluster. Once again unique to our approach, the LN operates as a normal node within the infrastructure in addition to its management responsibility towards the cluster. At the highest level, the Data Centre Controller (DC) manages the controllers one level below it.

Our scalable hybrid architecture, SHDF, attempts to service resource requests at the lowest local level possible, in order to reduce the overhead of servicing the request [39] and to reduce the performance impact of migrating VMs across cluster boundaries [4].

### 5.1 Controller Functionality - VM migration

When a node cannot satisfy the demands of the VMs it hosts, it starts an escalation process that aims to resolve the request at the lowest possible level. The MA running on the stressed node and the LN cooperate to resolve the escalated VM migration, by using our provided framework mechanisms.

The process starts within the NC's overlay by sending a request to other nodes within the same overlay (Step 1), shown in Figure 3, by using the accumulated metrics of other nodes. If a target node is available and accepts the migration request, then the migration process completes for this cycle. If the selected target node does not accept, other nodes within the overlay are attempted until no further options are available within the overlay. If a target is not available within the overlay, the stressed NC escalates the VM migration request to the LN (Step 2), and the MA running on the LN can query the cluster records from all the overlays, which have state data from all nodes in the cluster, to find a suitable node to house an escalated VM. If the LN locates a target within the cluster, the migration request is forwarded (Step 3). If the LN cannot find a suitable target for the migration within the cluster, it will use its knowledge of other available clusters, through participation in the LNs overlays, to forward the migration request to another LN (Step 4). This target LN will repeat the process performed by the forwarding LN, and attempt nodes within this cluster (Step 5). If a suitable target is found the process competes. If the LN in Step 5 cannot find a target, the request is rejected back to the forwarding LN, which will attempt other LNs in its overlay. If a suitable target is found through another LN, the process completes. If this fails to find a suitable target, the request is escalated to the DC Controllers (Step 6). The DC has a view of the entire infrastructure and can forward the request to other LNs (Step 7), which the original LN does not cooperate with. This recipient LN will repeat the process carried out by other LNs in the escalation chain (Step 8), and if a target is found the process completes. If a target is not found, the recent LN will reject the request back to the DC controller (Step 9). The DC controller will attempt other LNs, which have not been already tired, until a target is found or all LNs have been attempted.

If a target is not found then the infrastructure is highly stressed and the request is rejected back to the original escalating NC. In each of these escalation phases, the MA uses data from the data dissemination to decide on the list of targets to forward a request to. As requests progress through the escalation process, they are assigned an increasing priority, which can be used by the MA in the decision making process. For example, the MA may choose to prioritise finding a host for an escalated VM compared to a new VM placement.

### 5.2 Controller Functionality - Consolidation

At periodic time intervals and changes in utilisation, each of the management controllers and LNs can invoke a consolidation process where the MA can examine the state of the infrastructure and for every node under its management, decide to migrate some VMs from a node, migrate all VMs off a node and switch the node off or no change.

The advantage of SHDF is it allows the nodes to primarily operate in a distributed manner for time sensitive operations such as VM migrations, which could improve SLA violation metrics. In this paper, we focus on VM migration to achieve QoS metrics, and while RL consolidation is out of scope, we use a simple heuristic to perform regular consolidation.

## 6 Proposed Reinforcement Learning Management Algorithm

In our previous work [24], we have shown management algorithms (MAs) are widely covered in the literature, and drive the decision making process in cloud systems adaptation. The MA assigns resources in the infrastructure and regularly assesses the satisfaction of such assignments in achieving a given Service Level Agreement (SLA). The frequency of this assessment is influenced by the time complexity of the algorithm; the lower the complexity, the more frequently the algorithm can be executed. The approaches in the literature tend to be threshold-based [23], however, shared cloud environments have a significant uncertainty, and it is beneficial for the MA to be able to update the parameters of the decision making process to cope with a changing environment. With the promising results of applying RL in cloud resource management [19, 8, 10], we propose a RL based approach for VM migration, which builds on the MF from our earlier work, specifically to satisfy QoS metrics such as SLA. We build on SHDF by adding multiple modules that implement RL based agents at different levels of the SHDF hierarchy. On both the NC and LN, we add *monitoring, classification and learning modules* that provide the RL capability. The NC and LN perform their roles and escalation process on the hybrid architecture, as shown in Figure 3. The new modules and their operation are shown in Figure 4 and discussed below:

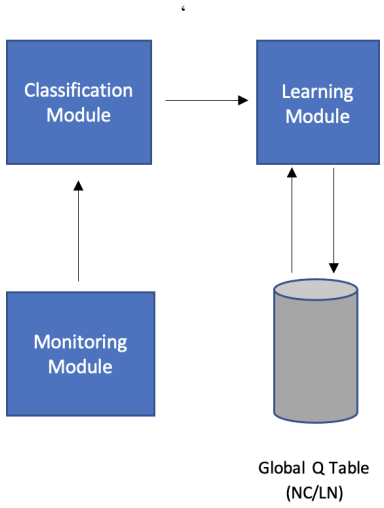



**Fig. 4** Proposed RL decision making steps

- A *Monitoring Module* tracks VM response times and is used as input by other modules to manage the node. The module additionally tracks the outcome of reinforcement learning actions.
- A *Classification module* assesses the state of a node and the VMs running on it, by using input from the Monitoring node. The module decides if a VM is stressed.
- A *Learning Module* uses the other modules as input to carry out decision making. When a VM is classed as stressed, the module determines the available actions and runs the Q-learning algorithm to decide the action to take. The module additionally executes

the action and invokes the monitoring module to determine the outcome of the action, calculates the reward for each taken action and updates the Q-table.

Algorithm 1 shows the operations in the RL agent. Each agent initialises value estimates to 0, and in each decision making cycle, the agent classifies the current state of the node, which identifies stressed VMs. Based on the current state of the node, the learning module determines all possible options by calling *getPossibleActions*. This uses the current overlay state in *state.targetCPUgroup[i]* to determine the available target groups of CPU utilisation from 0 to 90%. From the available options, the learning module will choose an action using an $\varepsilon$-greedy policy with respect to Q. This policy ensures that not all the agent's actions are greedy with respect to Q, and the agent will sometimes choose a random action, which enables a tradeoff between exploration and exploitation. Based on the chosen action, the learning module will then execute the action.

To speed up learning and speed of convergence in the Q-learning, we employ parallel learning [6]. Each agent learns from its experience of visited states, and learns the value of unvisited states from cooperating nodes in the SHDF architecture. A Q-table is shared among all the agents for each level in the SHDF hierarchy, and thus there is a shared Q-table for all NCs and a separate shared Q-table for all LNs. In addition to speeding up the learning and convergence, this approach enables a unique cooperation between NCs and LNs. When a NC escalates a migration to the LN in a given state, the reward from this action is tracked as part of the learning process.

**Algorithm 1** RL@NC

```
1:  procedure REGULARCHECK
2:      state ← classifyState(VM.getMonitoringData)
3:      actions ← state.getPossibleActions
4:      stateAction ← LearningModule.chooseAction(actions)
5:      switch stateAction do
6:          case powerNode
7:              result ← powerNewNode(vmToMigrate)
8:          case migrate
9:              result ← findOverlayNode(vmToMigrate, targetCPUWindow)
10:         case escalateToLead
11:             escalateToLead
12:         case noAction
13:             nop
14: procedure GETPOSSIBLEACTIONS(state)
15:     possibleActions ← null
16:     for targetCPUgroup[i] do
17:         if state.targetCPUgroup[i] > 0 then
18:             possibleActions.add ← targetCPUgroup[i]
19:     if offNodes > 0 then
20:         possibleActions.add ← powerOnNode
21:     possibleActions.add ← escalateToLead
22:
23: procedure CHOOSEACTION(possibleActions)
24:     if random < 1 − ε then
25:         action ← actionWithMaxQAtState(state, possibleActions)
26:     else
27:         action ← randomAction(possibleActions)
28:
```

### 6.1 RL State

The representation of the state is key to the RL decision making process. To overcome the state space dimensionality challenge with RL, we use a reduction approach and aggregate VMs to two states: Normal and Stressed. Response time has been used as a measure for application performance [18]. To account for variation in response time during the lifetime of an application, we use an approach similar to [43]. We apply linear regression on collected response time during each monitoring epoch, which by default is every two minutes. The classification module will deem a VM stressed when the 95th percentile of response time, during a monitoring period, is above a defined SLA threshold that by default is 500ms. We categorise the state of VMs as *Normal* when the 95th percentile of the response time is below the defined SLA level. The classification of state occurs during the regular node check, shown in line 2 of Algorithm 1.

### 6.2 RL Actions

Each node contains an RL agent, which carries out decision making. The RL agent carries out actions to achieve QoS metrics and balance this with energy consumption. To perform migration when a VM is stressed, the agent needs to identify a new target node to host the VM. The hybrid architecture provides an expanding set of options to migrate a VM, starting within the overlay where the VM resides and then onto the cluster and the rest of the data centre, a cooperation facilitated through the LN.

RL actions are contextual to the current state, and the agent ensures actions are valid by filtering non applicable actions, as shown in the getPossibleActions method in line 3 of Algorithm 1. For example, when the VM is in *Normal* state, no migration actions are available to the RL agent.

The goal for the RL agents is to find the actions that reduce SLA violations (maximise reward) when an agent enters a given state, shown in method *chooseAction* in line 4 of Algorithm 1. During the decision making process, the agent chooses between powering on a new node to house the migrating VM (line 6), migrating the VM to another target node within the same overlay (line 8), escalating to the Lead Node (line 10) and taking no action (line 12). In the migrate within overlay case, the RL agent needs to choose a target node to send a stressed VM to. However, a large data centre will have many thousands of nodes, and tracking an action reward for each individual target node will lead to a large set of actions and a large Q-table. To solve this, we simplify the RL action space and use a reduction mechanism that groups target nodes based on their CPU utilisation. By default we use 10 groups, 10% each using Equation 2, which creates target groups from 0 to 9. For example, action group1 means migrate the stressed VM to a node with an average CPU utilisation of 10% to 19%. Selecting action group6 means migrate to a node with CPU utilisation of 60% to 69%. Once an action is selected, we use a greedy policy to select the first available node that fits the action group. For example, an action of group6, will result in the first available node that meets the requirements of the VM and has an average utilisation between 60% and 69% to become selected as the target node. Based on CPU utilisation, we identify the target groups available and add these as options for the RL agent to choose from (lines 16 to 18). We additionally account for the number of available switched off nodes and add these as an option to the RL agent to switch on (lines 19 to 20). In most cases the agent chooses an action, from the available options, that maximises future reward (line 25). With a small

probability of $\varepsilon$, the agent will choose to explore and not exploit, by randomly selecting an available action (line 27).

$$targetGroup = \frac{avgCpuUtilization(node)}{actionGroups} \tag{2}$$

The NC can select actions to migrate a stressed VM within the overlay, as well escalate to a LN. We modify the original hybrid architecture, where escalation to the LN was only available when no target nodes were available in the overlay, to being able to escalate at any point using a RL action (line 12). The RL based migration and escalation process is shown in Figure 5. A LN deals with escalation of VM migrations from a NC, and performs RL actions within the scope of a whole cluster. Table 2 shows the RL actions carried out by a NC and Table 3 shows the LN actions. Similar to the original hybrid architecture, the process starts within the NC's overlay, where a NC uses the RL modules to make a migration. The difference in this new version, is the NC can choose to migrate within the overlay (Step 1) or make an escalation to the LN (Step 2), based on the RL agent and the actions from Table 2. If an escalation to the LN is chosen, the RL agent running on the LN will classify the cluster state and choose an action from Table 3, which can be within the cluster (Step 3) or a migration outside the cluster (Step 4). Once the action is executed, both the LN and NC RL agents receive a reward post action completion. The NC receives a reward for the escalation action (Step 5) and the LN for the choice of action within the cluster or outside it (Step 6).

Consolidation using RL is outside the scope of this work, and we use a simple heuristic to perform regular consolidation at the cluster level. The heuristic, based on [31], classifies all the nodes at the cluster level as *partially utilised, under utilised or empty*. All under utilised nodes become candidates for migrating VMs away from to other partly utilised nodes. When the last VM is migrated away from a node and it becomes empty, the node is switched off.

**Table 2** NC RL Actions

| Action | Description |
|---|---|
| Migrate [0,1,2,3,4,5,6,7,8,9] inside overlay | Migrate to target node with utilisation [1 to 9%, 10 to 19%, 20 to 29%, 30 to 39%, 40 to 49%, 50 to 59%, 60 to 69%, 70 to 79%, 80 to 89%, 90 to 99%] |
| Migrate Outside Overlay | Escalate migration to LN |
| Power Node inside overlay | Power on node & migrate |
| Do nothing | Do nothing |

**Table 3** LN RL Actions

| Action | Description |
|---|---|
| Migrate [0,1,2,3,4,5,6,7,8,9] inside cluster | Migrate to target node with utilisation [1 to 9%, 10 to 19%, 20 to 29%, 30 to 39%, 40 to 49%, 50 to 59%, 60 to 69%, 70 to 79%, 80 to 89%, 90 to 99%] |
| Migrate Outside cluster | Escalate migration to data centre controller |
| Power Node inside cluster | Power on node & migrate |

**Algorithm 2** Update Q-table

1: **procedure** UPDATEQ(action)
2: s' ← classifyState(VM.getMonitoringData)
3: responseTime ← monitoringModule.getRT(vm)
4: reward ← calculateReward(responseTime)
5: $Q(s,a) \leftarrow Q(s,a) + \alpha[reward + \gamma Q\_max(s',a) - Q(s,a)]$

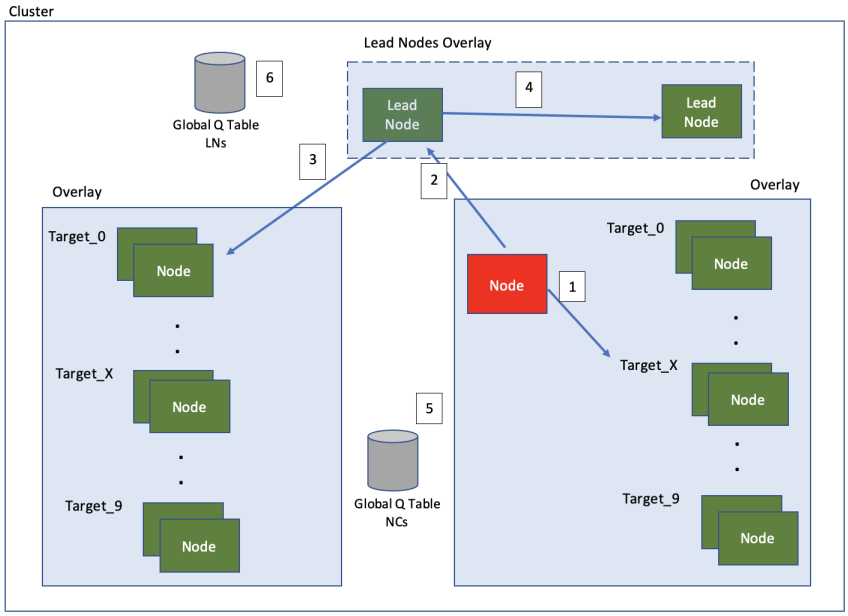



**Fig. 5** RL migration & escalation process

### 6.3 Reward

The goal for RL is to maximise rewards through incrementally mapping states to actions. The monitoring and classification modules regularly monitor a node and capture the state, which enables a subsequent action from the learning module. After an action is executed, there is a waiting period, which is action dependent. Boot up actions take a defined amount of time, typically set to 30 seconds, while migrations take a length of time that is dependent on the amount of memory used by the VM. After the action wait time, the monitoring module will calculate a reward for the action using Equation 3, and is shown in Algorithm 2. The update starts by classifying the current state of a VM (line 2), and calculating the archived reward based on Equation 3 in line 4. The Q-learning update is then applied based on Equation 1 in line 5. As our goal is to reduce SLA violations, and balance this with energy consumption, the reward function should reflect VM performance (yield) and resource utilisation (cost), and punish actions that degrade VM performance or significantly increase cost. Our reward function is shown in Equation 3, where yield is the gain in a QoS metric, and cost is the increase in energy consumption, represented by the delta in CPU utilisation. As nodes consume up to 70% of their full utilisation in idle state [13], the *Power on node*

action incurs a *penalty* of 0.7. Other actions do not carry a penalty.

$$Reward = yield - cost - penalty, \tag{3}$$

where $yield \in [-1,1]$ decaying to $-1$ for the worst QoS and 1 when QoS is met. Cost is the change in CPU utilisation where $cost \in [0,1]$. We use *Response Time* as our key QoS metric. Response Time ($rt$) represents the time it takes to execute a request to an application running inside a hosted VM, and reflects the CPU resources that are assigned to the VM. The monitoring module captures response time in the 95th percentile. When a VM is moved to a new target node, it will likely experience a change in $rt$; migration may also impact other VMs running on the target node. To capture this, our yield of $rt$ is calculated based on Equation 4, where $m$ is the number of all VMs running on the target node.

$$yield = \sum_{j=1}^{m} y(rt_j)\,. \tag{4}$$

For each VM, when $rt$ is a value below the $TargetRT$ and therefore satisfying SLA, the reward is 1. When $rt$ is above the $TargetRT$ for the VM, the function will punish actions that cause SLA violations, as shown in Equation 5.

$$y(rt) = \begin{cases} TargetRT - rt\ , rt > TargetRT, \\ 1 \qquad\qquad\quad rt < TargetRT \end{cases} \tag{5}$$

*Energy Consumption* is the second component used by the reward function, and captures the energy utilisation cost of the action, based on the CPU utilisation of the target node before and after the action. This value helps the learnt policy to move towards actions that balance meeting SLAs with energy consumption, and is shown in Equation 6.

$$cost = postActionUtilisation - preActionUtilisation \tag{6}$$

Each NC and LN carries out an action, receives a reward for the action, and updates the shared global NC and LN Q-tables respectively, as shown in Algorithm 2.

## 7 Experiment Setup and Evaluation

In this section, we evaluate different approaches and compare our RL based Management Algorithm (MA) with multiple systems from literature, on both the MA and Management Framework (MF) dimensions. We choose to compare to a heuristic [7] MA that uses a Modified Best Fit Decreasing approach to migrate a VM, which has been shown to be effective in practice. For MF, we choose the hybrid architecture from our earlier work [24] and a centralized MF, due to the popularity of centralized MFs in the literature, and we combine MFs and MAs with varying workloads and VM configurations. Thus, our evaluation set is:

- RL MA combined with Hybrid MF
- Heuristic MA combined with Hybrid MF
- RL MA combined with Centralized MF
- Heuristic MA combined Centralized MF

This combination enables us to evaluate the performance of the proposed RL MA, by comparing to a Heuristic MA, and investigate benefits that can be realised from combining the RL MA with the Hybrid MF.

We used simulation to facilitate rapid development of experiments of large data centres. We selected DCSim [57] because of its extensibility and existing implementation of a centralized architecture, allowing us to create baseline comparators for our proposed RL approach. We instantiate SHDF with three levels of controllers, running on the root of the data centre (DC Controller), the cluster manager (LN) and executing nodes (NC).

### 7.1 Simulator Setup

DCSim allows a VM to use more CPU than reserved, up to an amount that does not impact other VMs. Like [31], for the heuristic based approaches we use a CPU utilisation thresholds of 90% for high, indicating stress level, and we use 60% for low, indicating low utilisation.

In DCSim, an application is modelled as an interactive multi-tiered web application. Each application has a specified client think time, a workload component and a request service time, which is the amount of time required to process each incoming request. The workload defines the current number of clients connected to the application, which can change at discrete points in the simulation based on a *trace* file. The resource requirements are defined as its resource size, which is the expected amount of CPU, memory, bandwidth and storage. DCSim treats bandwidth and storage as fixed requirements, however, CPU requirements can be varied across the simulation based on the VM demands. DCSim applies a cost to VM migration including the time taken for migration, as a function of memory consumed by a VM, and factors in the bandwidth required for the VM migration on the hosting node. Additionally, the boot time of a switched off node has an elapsed time cost. The time taken to switch on a node for migration is reflected in the time period the VM is in a stressed state, and therefore the SLA achieved by a VM. Due to the complexities of building accurate power models, we focus our investigation on scalability metrics.

### 7.2 Workload and SLA Violations

We run the experiments at a load that requires more than 70% of the CPU resources of active nodes. Each simulated application contains a workload trace based on the number of incoming requests to web servers from publicly available traces; we used the following traces included with DCSim: Google 1, Google 3, EPA (Environment Protection Agency) and Clarknet. Each workload has an average normalized load of: 0.74, 0.31, 0.24 for Google 1, Google 3, Clarknet and EPA respectively. Figure 6 shows the normalized shape of the workload requests for each of these traces. DCSim divides traces into equal length segments, and total the number of requests in each interval. The values of each interval are then normalized to [0, 1], with 0 being zero requests, and 1 being the maximum number of requests received in an interval. These are then scaled to match the VM being used; for example if a VM uses one core at 2GHz, the normalized trace is scaled by 2000. We create VMs with different cores and RAM configurations, as shown in Table 4. Each experiment is run to simulate 24 hours, and when there is not enough trace data for a an experiment duration, we loop to the beginning of the trace.

### 7.3 Modelling the Impact of Decision Making

DCSim [57] applies a migration cost once a VM is selected for migration, by adding additional time to complete the migration based on the amount of memory used by the VM.

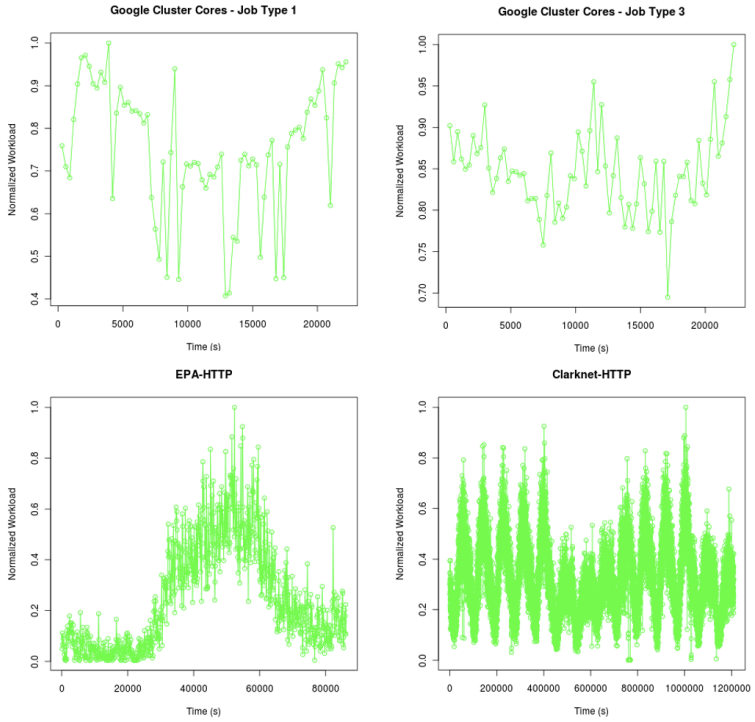



**Fig. 6** Original normalized traces used [57]

**Table 4** Experiment configuration

| Config | Config options | Base config |
|---|---|---|
| VM Core (Mhz) | 1000,1300,2500 | round robin [1000,1300,2500] |
| Node Capacity (Mhz) | 3000 | 3000 |
| Number of Cores | 1,2 | round robin [1,2] |
| VM Memory (MBs) | 1024,2048 | round robin [1024,2048] |
| Workload | clarknet, EPA, Google 1, Google 3 | round robin [clarknet, EPA, Google 1, Google 3] |
| Number of Nodes | 1000, 2000,3000, 4000, 5000 | 1000 |
| Node stress check frequency | 2 minutes | 2 minutes |
| VM service time | 0.2 seconds | 0.2 seconds |
| RL parameters | $\alpha$=0.1, $\gamma$=0.7, $\varepsilon$=0.1 | $\alpha$=0.1, $\gamma$=0.7, $\varepsilon$=0.1 |
| Target Response Time | 0.5 | 0.5 |

However, DCSim does not account for the time it takes to execute the decision making process, or the impact of such time. The length of the decision making process impacts stressed nodes by increasing the amount of time the node stays in a stressed state. In a centralized architecture, all nodes are used as input into the decision making process. Therefore, the execution of decision making could get progressively higher, as the number of managed nodes increases.

To capture the cost of the decision making, we extend DCSim to measure the amount of time during decision making, and add this time to the VM migration duration. As the

decision making execution time varies based on the MA and the hardware it is running on, we add a configurable scaling factor that can be applied to the measured execution time.

## 7.4 Data Centre

Our experiments use nodes modelled as ProLiant DL380 G5 Quad Core [1], with 2 dual-core 3GHz CPUs and 16GB of memory. The number of nodes used is specified in each of the experiments, with a minimum of 1,000 nodes. We assume that the data centre supports live VM migration, as this technique is currently supported by most major hypervisor technologies, such VMware [2] or Xen [14].

To minimise the number of variables in our experiments, we chose to keep a homogeneous infrastructure, with the same specification for all of the nodes. The various parameters used in our evaluation are outlined in Table 4.

## 7.5 Experiments

Our goal is to demonstrate improved QoS metrics for the RL MA, and we evaluate this by examining *SLA violations*. We evaluate our proposal under varying workloads and draw a comparison between our proposal and several related techniques.

Simulated applications are modelled as interactive web servers, running inside a VM, and an *SLA Violation* occurs when response time associated with the VM exceeds the *target response time*. We evaluate all architectures under varying scenarios to understand the impact on SLA violations. Initially, we evaluate a *mixed workload* scenario to represent the varying workloads deployed on data centres. To understand the impact of specific workloads, we then evaluate these individually. We additionally examine scalability and how the approaches cope with a varying arrival rate for new workloads.

### *7.5.1 No Adaptation*

In this experiment, we use a combined workload of all traces in a round robin approach, where each created VM uses a workload trace from all of the available traces. We use the centralized architecture with 1000 homogeneous nodes and no adaptation is invoked. When a node becomes stressed, it remains stressed for the reminder of the experiment, and exhibited 117045 instances of SLA violations.

### *7.5.2 Mixed Workload Assessment*

In this experiment, we use a combined workload of all traces in a round robin approach, where each created VM uses a different workload trace, in a deterministic order. The mixed workload simulates different workloads that could be experienced in a data centre. Workload arrival rate is the frequency that new VM placement requests arrive at the data centre. For this experiment we use an arrival rate of 180 new VMs per hour, with each VM running for 10 hours before shutting down. The experiment simulates 24 hours of elapsed time. We use 1000 nodes and the base configuration in Table 4. The SLA violation results are shown on Figure 7 .

The hybrid architecture achieved lower SLA violation compared to centralized, and the RL hybrid outperformed by 69.9%, 320.0% and 468.3% against the heuristic hybrid, RL

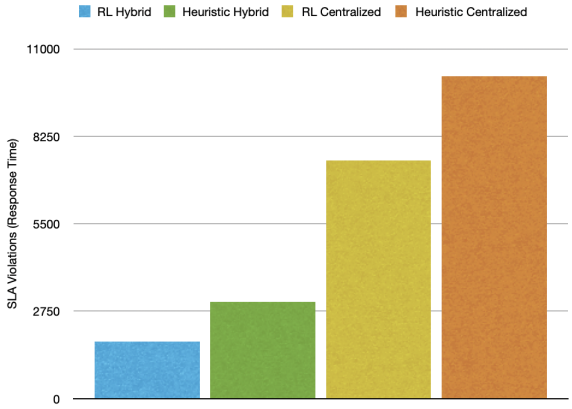



**Fig. 7** Mixed workload results: Number of SLA Violations response time $> 0.5$s

centralized and heuristic centralized, respectively. This is due to the RL approach of discovering target allocations that achieve the desired VM response time, and thus reduce SLA violations. The reward function drives behaviour of RL to choose target nodes that are switched on and have lower existing CPU utilisation. While the centralized architecture benefited from using RL, the hybrid architecture benefited more due to its autonomous approach to decision making, which provides rapid decision making, combined with a target node selection that maintains VM response time, thus reducing SLA violations.

Compared to the *No adaptation* experiment, all approaches unsurprisingly reduced SLA violations.

### 7.6 Workload Impact

To investigate the impact of workload further, we use the individual workloads from the previous experiment and evaluate them individually. For this experiment we use an arrival rate of 180 new VMs per hour, with each VM running for 10 hours before shutting down. The experiment simulated 24 hours of elapsed time. We use the base configuration in Table 4.

The results for the Google 1 workload are shown in Figure 8, and show the effect on SLA violations. This workload has an average of 0.74 normalized load and thus high stress. The RL hybrid approach outperformed other approaches by 119.4%, 248.4% and 299.7% against the heuristic hybrid, RL centralized and heuristic centralized respectively. The RL hybrid approach continues to perform well on this stressful workload, and shows further improvement in this workload against the heuristic hybrid; 119% versus 69% in the mixed workload, indicating the hybrid architecture benefits from the RL learnt policy, which favours target nodes that achieve target response time, while the heuristic approach does not take feedback signal on target node selection.

The results for the Google 3 workload are shown in Figure 9, and show the effect on SLA violations. This workload has an average of .83 normalized load and is the highest workload stress we have evaluated. The hybrid approach outperformed other approaches by 192.0%, 300.6% and 555.0% against the heuristic hybrid, RL centralized and heuristic centralized respectively. The RL hybrid approach performs even better on this workload,

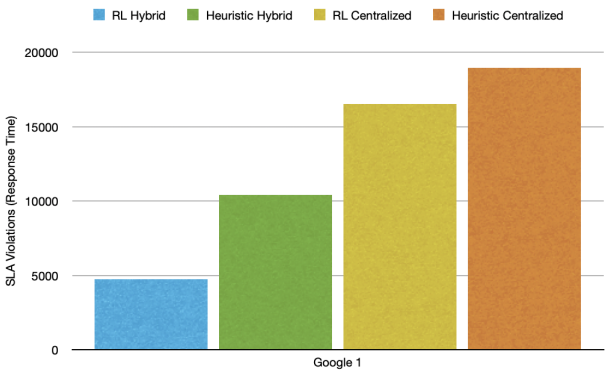



**Fig. 8** Google 1 workload results - Number of SLA Violations response time $>$ 0.5s

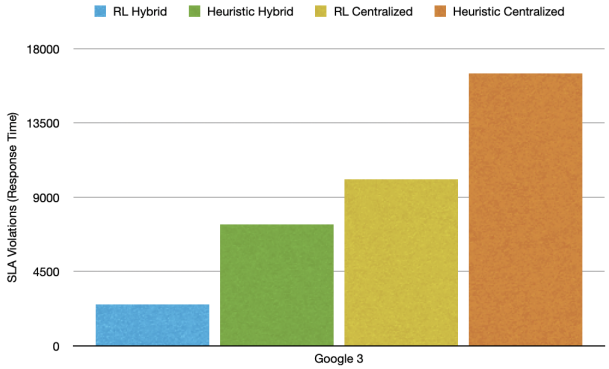



**Fig. 9** Google 3 workload results - Number of SLA Violations response time $>$ 0.5s

compared to the Google 1 workload, indicating that more stressful workloads benefit form the learned RL policy, which favours target node selection that meets target response time.

The results for the Clarknet workload are shown in Figure 10, and show the effect on SLA violations. This workload has an average of .31 normalized load and is a lower stress compared to the Google workloads. The hybrid approach outperformed other approaches by 85.7%, 109.9% and 156.6% against the heuristic hybrid, RL centralized and heuristic centralized, respectively. These are lower improvements, indicating higher stress workload benefit more from the RL hybrid approach.

The results for the EPA workload are shown in Figure 11, and show the effect on SLA violations. This workload has an average of .24 normalized load and is the least stress workload we have evaluated. The hybrid approach outperformed other approaches by 54.9%, 212.1% and 269.2% against the heuristic hybrid, RL centralized and heuristic centralized, respectively. These are lower improvements, indicating higher stress workloads benefit more from the RL hybrid approach.

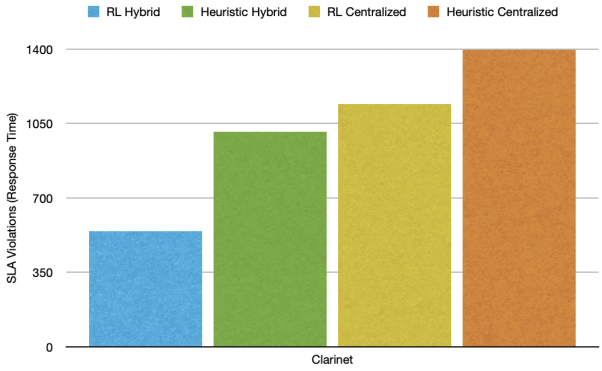



**Fig. 10** Clarknet workload results - Number of SLA Violations response time $> 0.5$s

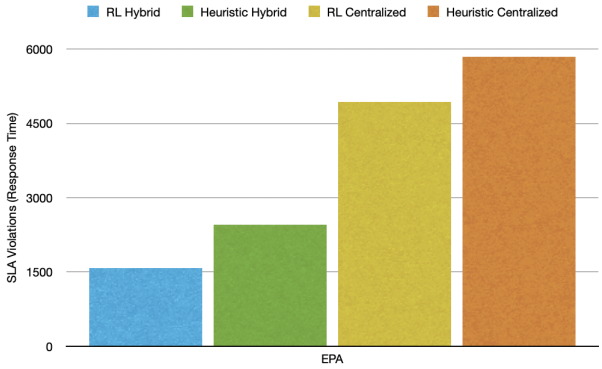
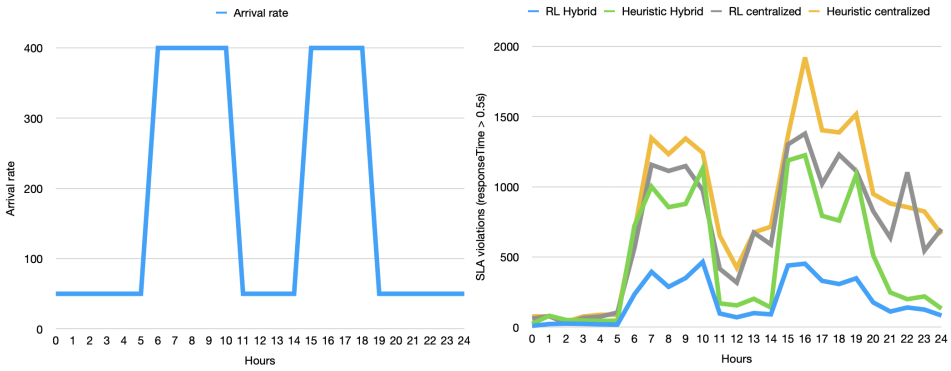



**Fig. 11** EPA workload results - Number of SLA Violations response time $> 0.5$s

### 7.7 Dynamic Workload

To evaluate how the different approaches cope with multiple high arrival rates, this experiment involves several sharp increases in arrival rate, by 400 VMs per hour each time. High arrival rate for VMs causes more VMs to be placed in the infrastructure, and in turn more VMs that can exhibit SLA violations. We examine the scenario in Figure 12a. There is a small and steady flow of new VM creation requests, followed by two sharp increases in number of VM creation requests that persist for several hours, followed by a return to the previous small steady number of VM creation requests. Each VM runs for 1 hour before shutting down, and we use the Google 3 trace. The experiment simulates 24 hours of elapsed time, and we use the base configuration in Table 4. The high arrival rate of VMs on the infrastructure, as each VM runs the Google 3 workload, increases the VMs that begin to experience a stress state where they do not deliver the requested CPU demand, and thus enter SLA violation. We hypothesised that this could be challenging for RL approaches, as they could learn suitable responses for arrival rates that are not sustained.

The results are shown in Figure 12b, and show number of incurred SLA violations. The hybrid based approaches outperform the centralized approach, due to the rapid decision making process. The RL hybrid outperforms the heuristic hybrid by receiving a reward for powering on new nodes as the load increases, shown in Figure 13. During each of the spikes




(a) (b)

**Fig. 12** High dynamic workload: a)Arrival rate patterns, b) SLA violations for dynamic arrival rate

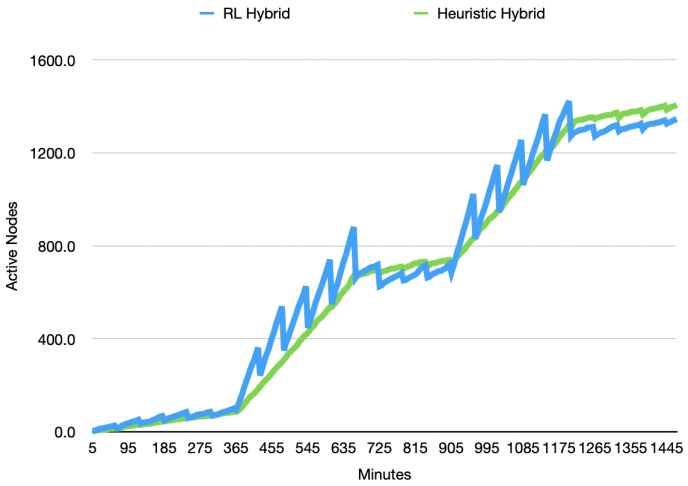



**Fig. 13** Active Nodes in RL Hybrid versus Heuristic Hybrid

at hours 5 and 14, the RL switched on more nodes versus steady new nodes in the heuristic hybrid. While the RL centralized has an improved target node allocation, compared to the heuristic centralized, it continued to suffer from the time taken for the decision making inherent in the centralized architecture.

## 7.8 Scalability

To evaluate how the approaches scale, we use the most stressful workload, Google 3, and maintain the stress ratio for each node, by increasing the load and the number of nodes in the data centre. Similar to other experiments, we simulate 24 hours and the results are shown in Table 5.

As the load ratio increases with more nodes, it is expected the number of SLA violations increases. The RL hybrid approach maintains its SLA performance as more nodes and VMs

**Table 5** Scalability of the different approaches - SLA Violations

| Number of Nodes | RL Hybrid | Heuristic Hybrid | RL Centralized | Heuristic Centralized |
|---|---|---|---|---|
| 1000 | 2518 | 7353 | 10257 | 13788 |
| 2000 | 5485 | 18817 | 22357 | 29262 |
| 3000 | 8537 | 29784 | 34161 | 44044 |
| 4000 | 11051 | 43633 | 46882 | 59026 |
| 5000 | 17771 | 56678 | 58730 | 75208 |

are added to the data centre, with lower SLA violations than all other approaches, on average 239.5%, 294.0% and 410.9% compared to the heuristic hybrid, RL centralized and heuristic centralized respectively.

### 7.9 Learning Performance

Convergence behaviour of an RL agent is a useful indicator to show whether the agent is learning an optimal policy. A preferable convergence shape is one where cumulative reward can gradually increase through time and converge to a high value [66]. While the RL hybrid and RL centralized use the same RL approach to monitor, classify and action node state, they have different decision making architectures. The RL hybrid performs parallel decentralized learning and the centralized has a single learning agent. The hybrid architecture has more rapid decision making and is able to execute more actions in a given time window, and is able to converge on a policy faster than the centralized approach. Figure 14 shows the cumulative value of RL actions for both approaches, from the Google 3 workload experiment. Initially there is low stress on the infrastructure, and the RL approach is making *No-operation* actions and both RL hybrid and RL centralized are receiving similar rewards. As the stress on the infrastructure increases, the RL hybrid is able to observe rewards from actions quicker than the RL centralized approach, and thus accumulate higher value during the experiment.

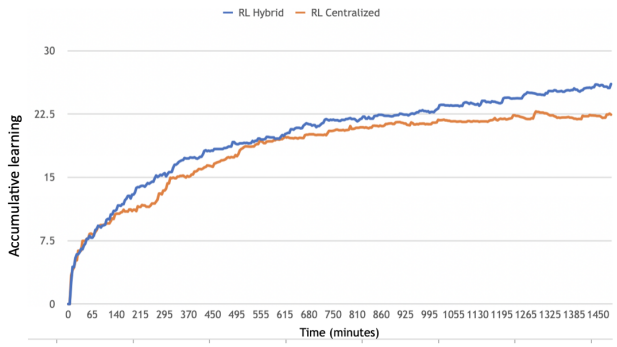



**Fig. 14** RL convergence: Hybrid versus Centralized

The unique cooperation between the RL agents running on NCs and LNs, enables escalation by the NC within one overlay to other overlays in the cluster. In this paper, we extended the original hybrid architecture, where escalation to the LN was only performed when no target nodes where available in the overlay, to NCs being able to escalate at any

point using an RL action. This enables the RL agent to explore escalation to LN prior to the overlay becoming full. Figure 15 shows the reduction in SLA violations in the RL escalation approach. By performing opportunistic escalation, the NC is reducing pressure on the overlay by leaving capacity in the overlay and thus reducing SLA violations.

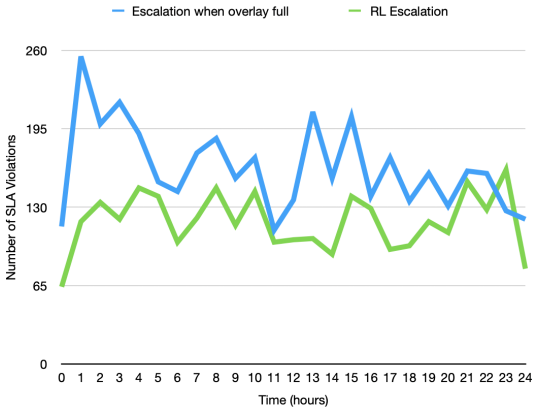



**Fig. 15** SLA violations: Hybrid Heuristic versus Hybrid RL with escalation

## 8 Conclusion

RL has shown great promise in a range of control applications. Using RL in managing cloud infrastructures offers adaptability and advantages over heuristic based approaches, which historically have been threshold based and require significant domain and application knowledge to define threshold values. In this paper, we presented a RL management algorithm that reduces the state and action space and uses a unique multi level RL agent cooperation, between a NC and LN in hierarchical management, to further improve SLA violations performance. This RL management algorithm integrates well into a hybrid management framework, from our earlier work. We evaluated the performance of our approach using workload traces and simulation, and compared the results obtained with an established heuristic, demonstrating significant improvement to SLA violations and high scalability. Future areas of improvement include expanding the RL state space, and enabling the RL MA to learn to migrate VMs just before they become stressed. Our RL approach can also be extended to include a RL approach for initial VM placement and consolidation. Additionally, it would be valuable to validate the simulation results with experimentation on actual cloud infrastructure.

## 9 Data Availability

The data traces used as input for this paper are part of the DCSim [57] simulator, and are available from Github: https://github.com/digs-uwo/dcsim/tree/master/traces. Most of the data generated by this research is contained in the results section of this paper, and the full datasets generated are available from the corresponding author on request.

## 10 Conflict of Interest

The authors declare that they have no conflict of interest.

## References


1. Hpe proliant (2016). URL https://www.hpe.com
2. Vmware. http://www.vmware.com (2016). URL http://www.vmware.com/
3. Ahmad, R.W., Gani, A., Hamid, S.H.A., Shiraz, M., Yousafzai, A., Xia, F.: A survey on virtual machine migration and server consolidation frameworks for cloud data centers. Journal of Network and Computer Applications **52**, 11–25 (2015). DOI doi.org/10.1016/j.jnca.2015.02.002
4. Aldhalaan, A., Menascé, D.A.: Autonomic allocation of communicating virtual machines in hierarchical cloud data centers. In: Cloud and Autonomic Computing (ICCAC), 2014 International Conference on, pp. 161–171. Cloud and Autonomic Computing (ICCAC), 2014 International Conference on (2014)
5. Arabnejad, H., Pahl, C., Jamshidi, P., Estrada, G.: A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 64–73 (2017). DOI 10.1109/CCGRID.2017.15
6. Barrett, E., Howley, E., Duggan, J.: Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurrency and Computation: Practice and Experience **25**(12), 1656–1674 (2013). DOI doi.org/10.1002/cpe.2864
7. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience **24**, 1397–1420 (2012)
8. Bibal Benifa, J.V., Dejey, D.: Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment. Mobile Networks and Applications **24**(4), 1348–1363 (2019). DOI 10.1007/s11036-018-0996-0
9. Bitsakos, C., Konstantinou, I., Koziris, N.: Derp: A deep reinforcement learning cloud system for elastic resource provisioning. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 21–29 (2018). DOI 10.1109/CloudCom2018.2018.00020
10. Bu, X., Rao, J., Xu, C.Z.: Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. IEEE Transactions on Parallel and Distributed Systems **24**(4), 681–690 (2013). DOI 10.1109/TPDS.2012.174
11. Calcavecchia, N.M., Caprarescu, B.A., Di Nitto, E., Dubois, D.J., Petcu, D.: Depas: a decentralized probabilistic algorithm for auto-scaling. Computing **94**(8), 701–730 (2012)
12. Chen, Z., Hu, J., Min, G.: Learning-based resource allocation in cloud data center using advantage actor-critic. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC), pp. 1–6 (2019). DOI 10.1109/ICC.2019.8761309
13. Chowdhury, M.R., Mahmud, M.R., Rahman, R.M.: Implementation and performance analysis of various vm placement strategies in cloudsim. Journal of Cloud Computing **4**(1), 20 (2015). DOI 10.1186/s13677-015-0045-5
14. Citrix: Xen. http://www.xenserver.org (2016). URL http://www.xenserver.org
15. Duggan, M., Flesk, K., Duggan, J., Howley, E., Barrett, E.: A reinforcement learning approach for dynamic selection of virtual machines in cloud data centres (2016). DOI 10.1109/INTECH.2016.7845053
16. Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S., Hester, T.: Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning (2021). DOI 10.1007/s10994-021-05961-4
17. Gahlawat, M., Sharma, P.: Survey of virtual machine placement in federated clouds. In: 2014 IEEE International Advance Computing Conference (IACC), pp. 735–738 (2014). DOI 10.1109/IAdCC.2014.6779415
18. Ghanbari, H., Simmons, B., Litoiu, M., Barna, C., Iszlai, G.: Optimal autoscaling in a iaas cloud. In: Proceedings of the 9th International Conference on Autonomic Computing, ICAC '12, pp. 173–178. Association for Computing Machinery, New York, NY, USA (2012). DOI 10.1145/2371536.2371567
19. Ghobaei-Arani, M., Jabbehdari, S., Pourmina, M.A.: An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. Future Generation Computer Systems **78**, 191 – 210 (2018). DOI doi.org/10.1016/j.future.2017.02.022
20. Gholipour, N., Arianyan, E., Buyya, R.: A novel energy-aware resource management technique using joint vm and container consolidation approach for green computing in cloud data centers. Simulation Modelling Practice and Theory **104**, 102127 (2020). DOI doi.org/10.1016/j.simpat.2020.102127

21. Guo, W., Tian, W., Ye, Y., Xu, L., Wu, K.: Cloud resource scheduling with deep reinforcement learning and imitation learning. IEEE Internet of Things Journal **8**(5), 3576–3586 (2021). DOI 10.1109/JIOT.2020.3025015
22. Gupta, M.K., Amgoth, T.: Resource-aware virtual machine placement algorithm for iaas cloud. The Journal of Supercomputing **74**(1), 122–140 (2018). DOI 10.1007/s11227-017-2112-9
23. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Adaptation in cloud resource configuration: a survey. Journal of Cloud Computing **5**(1), 1–16 (2016)
24. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Shdf - a scalable hierarchical distributed framework for data centre management. In: 2017 16th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 102–111. 16th International Symposium on Parallel and Distributed Computing (ISPDC) (2017). DOI 10.1109/ISPDC.2017.15
25. Jamshidi, P., Sharifloo, A.M., Pahl, C., Metzger, A., Estrada, G.: Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution. In: 2015 International Conference on Cloud and Autonomic Computing, pp. 208–211 (2015). DOI 10.1109/ICCAC.2015.35
26. Jangiti, S., Sriram. V.S., S.: Scalable and direct vector bin-packing heuristic based on residual resource ratios for virtual machine placement in cloud data centers. Computers & Electrical Engineering **68**, 44–61 (2018). DOI doi.org/10.1016/j.compeleceng.2018.03.029
27. Jauro, F., Chiroma, H., Gital, A.Y., Almutairi, M., Abdulhamid, S.M., Abawajy, J.H.: Deep learning architectures in emerging cloud computing architectures: Recent development, challenges and next research trend. Applied Soft Computing **96**, 106582 (2020). DOI doi.org/10.1016/j.asoc.2020.106582
28. Jin, Y., Bouzid, M., Kostadinov, D., Aghasaryan, A.: Resource management of cloud-enabled systems using model-free reinforcement learning. Annals of Telecommunications **74**(9), 625–636 (2019). DOI 10.1007/s12243-019-00720-y
29. John, I., Sreekantan, A., Bhatnagar, S.: Efficient adaptive resource provisioning for cloud applications using reinforcement learning. In: 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 271–272 (2019). DOI 10.1109/FAS-W.2019.00077
30. Kardani-Moghaddam, S., Buyya, R., Ramamohanarao, K.: Adrl: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds. IEEE Transactions on Parallel and Distributed Systems **32**(3), 514–526 (2021). DOI 10.1109/TPDS.2020.3025914
31. Keller, G., Tighe, M., Lutfiyya, H., Bauer, M.: A hierarchical, topology-aware approach to dynamic data centre management. In: Network Operations and Management Symposium (NOMS), pp. 1 –7. Network Operations and Management Symposium (NOMS) (2014)
32. Khan, T., Tian, W., Buyya, R.: Machine learning (ml)-centric resource management in cloud computing: A review and future directions (2021)
33. Kim, S., Choi, Y.r.: Constraint-aware vm placement in heterogeneous computing clusters. Cluster Computing **23**(1), 71–85 (2020). DOI 10.1007/s10586-019-02966-6
34. Lebre, A., Pastor, J., Simonet, A., Südholt, M.: Putting the next 500 vm placement algorithms to the acid test: The infrastructure provider viewpoint. IEEE Transactions on Parallel and Distributed Systems **30**(1), 204–217 (2019). DOI 10.1109/TPDS.2018.2855158
35. Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J., Wang, Y.: A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 372–382 (2017). DOI 10.1109/ICDCS.2017.123
36. Lolos, K., Konstantinou, I., Kantere, V., Koziris, N.: Elastic management of cloud applications using adaptive reinforcement learning. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 203–212 (2017). DOI 10.1109/BigData.2017.8257928
37. Masdari, M., Zangakani, M.: Green cloud computing using proactive virtual machine placement: Challenges and issues. Journal of Grid Computing **18**(4), 727–759 (2020). DOI 10.1007/s10723-019-09489-9
38. Matignon, L., Laurent, G.J., Fort-piat, N.L.: Improving reinforcement learning speed for robot control. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3172–3177 (2006). DOI 10.1109/IROS.2006.282341
39. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for cloud infrastructure management. Future Generation Computer Systems **29**(2), 472–487 (2013)
40. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). DOI 10.1038/nature14236
41. Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M.: Efficient resource provisioning for elastic cloud services based on machine learning techniques. Journal of Cloud Computing **8**(1), 5 (2019). DOI 10.1186/s13677-019-0128-9

42. Muller-Merbach, H.: Heuristics and their design: a survey. European Journal of Operational Research **8**(1), 1–23 (1981). URL https://ideas.repec.org/a/eee/ejores/v8y1981i1p1-23.html
43. Nouri, S.M.R., Li, H., Venugopal, S., Guo, W., He, M., Tian, W.: Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. Future Generation Computer Systems **94**, 765 – 780 (2019). DOI doi.org/10.1016/j.future.2018.11.049
44. Pantazoglou, M., Tzortzakis, G., Delis, A.: Decentralized and energy-efficient workload management in enterprise clouds. IEEE Transactions on Cloud Computing **4**(2), 196–209 (2016)
45. Pietri, I., Sakellariou, R.: Mapping virtual machines onto physical machines in cloud computing: A survey. ACM Comput. Surv. **49**(3) (2016). DOI 10.1145/2983575
46. Rao, J., Bu, X., Xu, C.Z., Wang, K.: A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In: 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 45–54 (2011). DOI 10.1109/MASCOTS.2011.47
47. Ren, H., Wang, Y., Xu, C., Chen, X.: Smig-rl: An evolutionary migration framework for cloud services based on deep reinforcement learning. ACM Trans. Internet Technol. **20**(4) (2020). DOI 10.1145/3414840
48. Sedaghat, M., Hernández-Rodriguez, F., Elmroth, E., Girdzijauskas, S.: Divide the task, multiply the outcome: Cooperative vm consolidation. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 300–305. IEEE International Conference on Cloud Computing Technology and Science, IEEE, Washington, DC, USA (2014)
49. Shaw, R., Howley, E., Barrett, E.: Applying reinforcement learning towards automating energy efficient virtual machine consolidation in cloud data centers. Information Systems p. 101722 (2021). DOI doi.org/10.1016/j.is.2021.101722
50. Silva Filho, M.C., Monteiro, C.C., Inácio, P.R., Freire, M.M.: Approaches for optimizing virtual machine placement and migration in cloud environments: A survey. Journal of Parallel and Distributed Computing **111**, 222–250 (2018). DOI doi.org/10.1016/j.jpdc.2017.08.010
51. Sina, M., Dehghan, M., Rahmani, A.M.: Car-plive: Cloud-assisted reinforcement learning based p2p live video streaming: a hybrid approach. Multimedia Tools and Applications **78**(23), 34095–34127 (2019). DOI 10.1007/s11042-019-08102-1
52. Sniezynski, B., Nawrocki, P., Wilk, M., Jarzab, M., Zielinski, K.: Vm reservation plan adaptation using machine learning in cloud computing. Journal of Grid Computing **17**(4), 797–812 (2019). DOI 10.1007/s10723-019-09487-x
53. Song, B., Hassan, M., Huh, E.n.: A novel heuristic-based task selection and allocation framework in dynamic collaborative cloud service platform. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science, pp. 360–367 (2010). DOI 10.1109/CloudCom.2010.53
54. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. Cambridge:MIT press (1998)
55. Thanh Binh, H.T., Phi Le, N., Minh, N.B., Thu Hai, T., Minh, N.Q., Bao Son, D.: A reinforcement learning algorithm for resource provisioning in mobile edge computing network. In: 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1–7 (2020). DOI 10.1109/IJCNN48605.2020.9206947
56. Tighe, M., Keller, G., Bauer, M., Lutfiyya: A distributed approach to dynamic vm management. In: Proceedings of the 9th International Conference on Network and Service Management, p. 166 to 170. Proceedings of the 9th International Conference on Network and Service Management (2013)
57. Tighe, M., Keller, G., Bauer, M., Lutfiyya, H.: Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management. In: Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), pp. 385–392. Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm) (2012)
58. Walsh, W., Tesauro, G., Kephart, J., Das, R.: Utility functions in autonomic systems. In: International Conference on Autonomic Computing, 2004. Proceedings., pp. 70–77 (2004). DOI 10.1109/ICAC.2004.1301349
59. Watkins, C.J.C.H.: Learning from delayed rewards. In: Ph.D. Thesis, (1989)
60. Witanto, J.N., Lim, H., Atiquzzaman, M.: Adaptive selection of dynamic vm consolidation algorithm using neural network for cloud resource management. Future Generation Computer Systems **87**, 35–42 (2018). DOI doi.org/10.1016/j.future.2018.04.075
61. Wu, Y., Tang, M., Fraser, W.: A simulated annealing algorithm for energy efficient virtual machine placement. In: 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 1245–1250 (2012). DOI 10.1109/ICSMC.2012.6377903
62. Wuhib, F., Stadler, R., Spreitzer, M.: Dynamic resource allocation with management objectives: implementation for an openstack cloud. IEEE Transactions on Network and Service Management **9**(2), 213–225 (2012)

63. Xu, H., Liu, Y., Wei, W., Xue, Y.: Migration cost and energy-aware virtual machine consolidation under cloud environments considering remaining runtime. International Journal of Parallel Programming **47**(3), 481–501 (2019). DOI 10.1007/s10766-018-00622-x
64. Yadav, M.P., Rohit, Yadav, D.K.: Resource provisioning through machine learning in cloud services. Arabian Journal for Science and Engineering (2021). DOI 10.1007/s13369-021-05864-5
65. Yadav, R., Zhang, W., Li, K., Liu, C., Shafiq, M., Karn, N.K.: An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center. Wireless Networks **26**(3), 1905–1919 (2020). DOI 10.1007/s11276-018-1874-1
66. Ying, C., Li, B., Ke, X., Guo, L.: Raven: Scheduling virtual machine migration during datacenter upgrades with reinforcement learning. Mobile Networks and Applications (2020). DOI 10.1007/s11036-020-01632-1
67. Zolfaghari, R., Sahafi, A., Rahmani, A.M., Rezaei, R.: Application of virtual machine consolidation in cloud computing systems. Sustainable Computing: Informatics and Systems **30**, 100524 (2021). DOI doi.org/10.1016/j.suscom.2021.100524

## 4.5 Dynamic Threshold Setting for VM Migration

Abdul R Hummaida, Norman W Paton and Rizos Sakellariou

**Publishing state:** Published in ESOCC 2022.

**Summary**: Invocation of VM migration is widely studied in the literature and typically uses dynamic CPU utilization threshold. One disadvantage of proposed approaches is they use node metrics (e.g., CPU and memory), and do not factor in VM performance metrics. This may lead to either early or delayed migration of VMs. We present a dynamic approach that factors the performance of a VM in setting the node's CPU threshold to use for migration.

We hypothesise that a fine-grained tracking of CPU utilization between 90% and 100% will enable our RL approach to detect a more optimal migration threshold, compared to a single group for the 90%+ CPU utilization, thus delaying the migration point. We apply the fine- grained approach on 90% and above to avoid an increase in the RL state space, which could impact convergence speed.

The Q-learning algorithm has some variables that may influence its performance, the learning rate $\alpha$ and the discount factor $\gamma$. The learning rate determines how the agent learns from recent updates. Values closer to zero make the agent exploit prior knowledge, and values closer to 1 make the agent consider the most recent information. The discount factor is used to dampen the reward's effect on the agent's choice of action and values closer to one will make the agent favour a long term reward.

In our experiments, we have observed some sensitivity to the chosen values for $\alpha$ and $\gamma$. Values closer to 1 for $\alpha$ result in higher errors in estimates and an increase in SLA violations. This is accompanied by some reduction in energy consumption. Given these results, we have chosen to use a value of 0.5 for $\alpha$, ensuring errors in the estimate are backed up. Values closer to 1 for $\gamma$ result in the agent favouring long term reward and delaying the migration point, resulting in more VMs being packed onto a node and lower energy consumption. This is also accompanied by an increase in SLA violations. Given the results of our experiments, we have chosen to use a value of 0.7 for $\gamma$ that balances SLA violations and energy consumption.

**Key contributions:** Contribution 5 (see Section 1.4).

# Dynamic Threshold Setting for VM Migration

Abdul Rahman Hummaida[0000−0002−3007−2289], Norman W Paton[0000−0003−2008−6617], and Rizos Sakellariou[0000−0002−6104−6649]

University of Manchester, Department of Computer Science, Kilburn Building, Oxford Rd, Manchester M13 9PL, UK
abdul.hummaida@postgrad.manchester.ac.uk,
{norman.paton,rizos}@manchester.ac.uk

**Abstract.** Cloud data centres require efficient management of resources and robust methods that consider SLA violations, node utilisation and simplify the adaptation decision making process. Thus resource management should be ideally solved in an online manner. To address this, approaches have been presented in the literature to set thresholds that trigger VM migration. One challenge with these approaches is they typically use node metrics (e.g., CPU and memory) as an indicator of VM performance and do not factor in VM performance metrics when setting the CPU migration threshold. A hypothesis is that migrating VMs without factoring in VM performance metrics, e.g., response time can lead to either early or delayed migration of VMs. We present an approach to discover the CPU utilization level for VM migration dynamically. This approach monitors VM response time and discovers the CPU threshold where response time would increase beyond a defined SLA level and uses this threshold for VM migration. We use reinforcement learning (RL) to learn when it is rewarding to migrate a VM. The RL reward function drives a policy towards high CPU utilisation and attaches a penalty to overachieving SLAs. We use simulation to evaluate the approach and compare it to 4 heuristics: Static, Interquartile Range, Median Absolute Deviation, Local Regression. The results show a significant reduction in SLA violations and an increase in CPU utilization of active nodes.

**Keywords:** Dynamic CPU threshold · Reinforcement Learning · VM Migration threshold.

## 1 Introduction

Platform as a service (PaaS) is a service model where Cloud Providers (CPs) provide hardware, software stacks and runtime environments for application development. Customers have control over the development environment, including configuration. CPs host the hardware and software on its infrastructure and remove the need for customers to maintain the application stack, runtime environments, operating systems and databases. To provide high levels of availability and reliability, CPs need to adapt the infrastructure regularly, which is typically comprised of VMs.

However, the VM migration process can be expensive, and thus there is a need to balance the benefit with the cost of the migration. This raises the challenge of deciding when VM migration should be invoked to achieve this balance. The constituent parts of VM migration include: (i) node overload detection, (ii) VM selection for migration from the overloaded node, and (iii) VM placement on a different target node. This is shown in Figure 1. From our earlier work [19], we have assumed that a Management Algorithm (MA) is responsible for deciding how incoming workloads are assigned to infrastructure resources by regularly assessing the satisfaction of such assignments in achieving a given SLA. The time complexity of the MA influences the frequency of this assessment; the lower the complexity, the more frequently the algorithm can be executed. The Management Framework (MF) enables the MA to execute by providing standard functionality, such as hierarchy level management, the scope of the infrastructure under control or aggregation of utilisation metrics. The combined functionality of the MA and MF results in workloads executing on infrastructure nodes and dynamic reassignment of workloads to resources. In this paper, we focus on node overload detection.

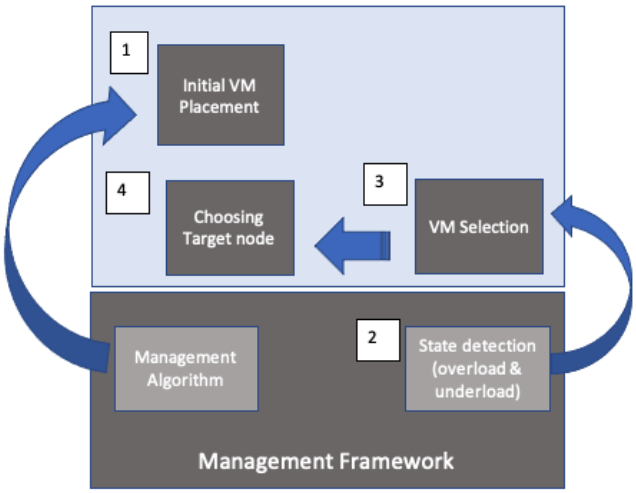



**Fig. 1.** Cloud Resource Management process

The overload detection methods used in the literature fall onto *reactive*, *proactive* and *hybrid* engagement [20]. Reactive approaches [30, 14, 2] invoke adaptation when a monitored metric, e.g. CPU utilisation, reaches a specific threshold or when an event is received, such as new VM placement or termination request. Proactive approaches [24, 5, 33] predict what demands will be

placed on the infrastructure and invoke adaptation ahead of the expected resource contention point. Hybrid approaches [22, 4] utilise proactive methods and combine these with reactive methods to engage adaptation for long and short term time scales.

The challenge we focus on is in reactive approaches, and these typically use *ad hoc* manually determined policies, such as threshold-based that are popular due to their simplicity. A key element to the threshold-based approach is the assumption there is a high chance that an overload occurs when a node's utilization exceeds the set threshold [1, 25, 3, 15, 39, 29]. Thus, the threshold level creates an association between a node metric, e.g. CPU utilization, and SLA violation. However, the threshold where SLA violations can occur varies based on the application and the node configuration. Creating a single threshold for all applications and node configurations is incredibly difficult [13]. While the current approaches can reduce overload, they can limit the utilization gains that can be achieved as they leave unused slack for each node. Additionally, threshold approaches can trigger unnecessary migrations as exceeding the set threshold does not necessarily equate to an SLA violation [12]. In addition to heuristics, other techniques have been used for node overload detection; [36] propose a multiobjective optimization that considers the CPU and memory utilization of VMs and nodes. The authors in [37] propose a bio-inspired method based on node susceptibility for host overload detection, and the authors in [29] propose a classical control theory approach.

Application performance is a measure of how well a service performs, and the metrics for measuring this include response time [18, 16]. Several CPs have monitoring services, including AzureWatch from Microsoft and CloudWatch from Amazon, that enable monitoring of VM performance hosted on their computing and storage cloud services [16]. We focus on cases where the response time of web-based applications forms part of the SLA between the customer and the CP. The response time can be measured and reported on using the CPs monitoring services.

We hypothesise that including VM performance in the migration decision making will lower the number of SLA violations. In this paper, we incorporate VM response time in a dynamic threshold detection approach and use RL to detect a rewarding threshold level to use by receiving feedback from the VM migration process. The main contributions of this paper are the following:

1. A coordinated migration method that automates the setting of CPU threshold, achieving lower SLA violations and increasing node utilization.
2. A model-free reinforcement learning algorithm for online VM migration that incorporates VM response time in decision making and removes human knowledge to set CPU threshold.
3. Evaluation of the proposed approach using simulation, appropriate workloads and a performance comparison against other approaches in the literature.

The rest of this paper is organised as follows. Section 2 describes related work. Section 3 describes the proposed reinforcement learning algorithm. Section

4 presents an evaluation of our implementation and compares it to four other heuristic dynamic threshold approaches. In Section 5, we conclude and discuss future work.

## 2 Related Work

Reactive approaches are typically implemented using threshold techniques [12] by triggering adaptation when a node's utilization reaches a given level. Beloglazov and Buyya [9] proposed a collection of adaptive policies for setting the upper thresholds: Interquartile Range, Median Absolute Deviation, Local Regression, and Robust Local Regression. The thresholds can be calculated through statistical analysis of historical node utilisation metrics. Other approaches include adaptive heuristic algorithms [39]. The authors in [26] proposed an overload and underloaded node detection. A node is deemed overloaded if the actual and the predicted total CPU usage of 7-time intervals ahead exceed the defined overload threshold. The authors in [25] propose multiple exponential weighted moving average algorithms to detect overloaded nodes. The authors also incorporated a probabilistic approach to counter the uncertainty of the long-term predictions and the cost of applying the VM migration. Other proposals include a regression-based algorithm to create an upper threshold for detecting overload [39]. The approach automatically adjusts the upper CPU utilization threshold based on the historical CPU utilization of the nodes. The authors in [15] use three upper CPU utilization thresholds that are set dynamically based on the conditions of CPU utilization. Other approaches attempted to create a composite metric for overload detection, that combines additional metrics to CPU utilization, such as memory and network BW utilization [1].

However, these approaches did not incorporate VM performance, such as response time in setting the node CPU threshold.

Cloud environments are dynamic and exhibit regular changes in the structure of workloads and access patterns. Aptly, Reinforcement learning (RL) can operate online, learn dynamically from interacting with a changing environment, and use new information to enhance decision making. RL approaches do not require prior knowledge of the optimization model and are not coded explicit instructions relating to which action to take next; instead, they learn actions through feedback from the environment. These features make RL well suited to cloud resource management [27]. RL is utilized in multiple approaches related to cloud resource management [6, 10, 28, 31], and here we focus on some of the approaches in the literature that use RL to reduce the complexity of setting adaptation thresholds.

The authors in [23] aim to remove the need for human knowledge to define adaptation rules by using using a fuzzy rule-based RL algorithm that learns and modifies fuzzy rules at runtime. The author's approach combines Q-learning, an RL algorithm, with fuzzy control where the fuzzy control facilitates human reasoning and the Q-learning allows dynamic rules adjustment. The authors in [7] also aim to adapt the configuration of an application dynamically. They propose

to use RL to manage threshold-based rules, where one controller modifies an application configuration, and another monitors the adaptation reward. In contrast to the approach in [23], the author's approach requires human knowledge to initialise the rules.

The authors in [32] propose to manage VM resource configurations by monitoring performance feedback from each VM. The authors aim to optimize the VM performance by learning the VM resource allocation that enhances metrics such as VM response time and throughput by using RL. The reconfiguration process happens periodically on a predefined time interval. A controller fetches the VMs current state and computes valid actions. The RL state is defined as a composite of VM memory size, scheduler credit and the number of virtual CPUs assigned to each VM. The RL method chooses an action and monitors the reward. Actions adjust resources such as the CPU and memory assigned to a VM. The work in [11] proposes CoTuner, for coordinated configuration of VM resources and parameters of their applications. Each VM has an agent that monitors the VM and adapts its configuration to the environment. Reconfiguration actions take place periodically at predefined time intervals. The RL method receives performance feedback and updates the VM and application configuration. For VM configuration, CoTuner can adjust both CPU and memory VM assignments. For applications, CoTuner can change parameter settings.

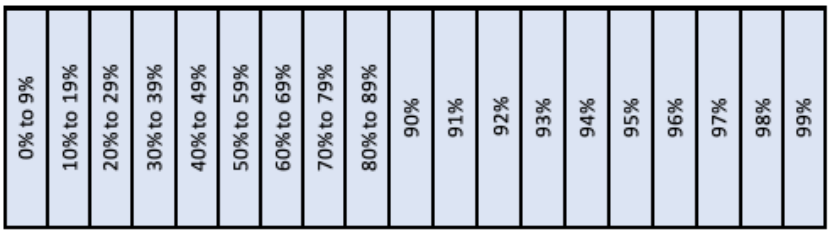
Similar to our proposed approach, the discussed methods use RL to dynamically change a threshold configuration to optimise performance and reduce SLA violations. In contrast, our approach uses a reduced RL state instead of tracking each VM CPU and memory configuration. We track the node CPU utilization as our primary RL state, resulting in a smaller RL state space and faster convergence compared to approaches with a more dense RL state.

This paper develops an RL-based controller to solve the challenge of determining a node CPU utilization threshold for VM migration and combine Q-Learning with an aggregated state action space to address the curse of dimensionality in Q-learning. We focus on node overload detection and aim to find the CPU utilization at which VM response time will start to degrade beyond a defined SLA target.

## 3 Proposed Reinforcement Learning Management Algorithm

In our previous work [19], we presented a novel hybrid hierarchical decentralized management framework that rapidly provides the information needed for scale decision making. In this hybrid architecture, higher-level controllers assist lower decentralised controllers. The lowest level controller manages a single node and enables it to be completely autonomous and cooperate with other autonomous nodes to facilitate VM migration. Nodes can receive escalation requests from higher-level controllers to accommodate a migration, and each node can choose to accept or reject these requests.

In this paper, we add an agent that implements our RL approach to each node in the infrastructure and combines this with our previously proposed hybrid hierarchical architecture. This creates RL agents that are both autonomous and cooperate in managing the data centre infrastructure.




(a)

**Fig. 2.** Granular CPU utilization state

### 3.1 State

Our goal is to address the challenge of setting a CPU threshold to invoke VM migration. We aim to regularly discover the node CPU utilization that returns the highest reward for performing a VM migration. To achieve this, we need to track nodes CPU state and associate a reward for migrating at each CPU state. However, the granularity of capturing the CPU state is crucial in avoiding the high dimensionality challenge in RL.

We use a state reduction approach and aggregate node state to groups, with each group based on their CPU utilisation, as shown in Figure 2. By default, we start by creating ten groups, 10% each, using Equation 1, which creates groups from 0 to 9. For example, State1 means the node state has an average CPU utilisation of 10% to 19%. State6 means the node has an average utilisation between 60% and 69%.

$$stateGroup = \frac{avgCpuUtilization(node)}{stateGroups} \tag{1}$$

We hypothesise that a fine-grained tracking of CPU utilization between 90% and 100% will enable our RL approach to detect a more optimal migration threshold, compared to a single group for the 90%+ CPU utilization. We apply the fine-grained approach on 90% and above to avoid an increase in the RL state space, which could impact convergence speed.

Periodically, each node additionally classifies the state for all running VMs as *Normal* or *Stressed*, and we use response time as a measure for application performance [17]. To account for variation in response time during the lifetime of an application, we use an approach similar to [28] and apply linear regression

on collected response time during each monitoring period. A VM is classed as stressed when the 95th percentile of response time during a monitoring period is above a defined SLA threshold that by default is 500ms. We categorise the state of VMs as *Normal* when the 95th percentile of the response time is below the defined SLA level. The classification of state occurs during the regular node check. When a VM is stressed, the RL agent always engages the migration mechanism and chooses an action, using the method described in the next section. When VMs are in a normal state, the agent will choose an action using an $\epsilon$-greedy policy [34] to decide if migration should be performed on this node state. This means with a small probability of $\epsilon$, the agent will choose to explore and not exploit by randomly selecting an available action. This leads the agent to learn the node CPU utilization with the highest reward for migration, which the agent exploits in future cycles.

### 3.2 Actions

Each node contains an RL agent in our architecture that carries out decision making. The RL agent performs actions to achieve QoS metrics and increase infrastructure utilization. As part of the decision making process, an RL agent needs to identify a new target node for the VM being migrated. The RL agent aims to find the actions that maximise reward and chooses a target node based on a CPU utilization group 0 to 9, based on [21]. When the RL agent chooses an action target2, it means migrating the VM to a node with CPU utilisation of 20% to 29%. Once an action is selected, we use a greedy policy to select the first available node that fits the action group. Typically, the agent chooses an action that maximises future reward from the available actions. The agent receives a reward after each action, which is described in the following section. This reward is used to update the node's state-action value pair using Equation 2 [38], where $\alpha$ is the learning rate and determines how the agent learns from recent updates. $\gamma$ is the discount factor used to dampen the reward's effect on the agent's choice of action. $MaxQ(s_{t+1}, a_{t_{+1}})$ returns the maximum estimate for the future state-action pair.

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma MaxQ((s_{t+1}, a_{t_{+1}}) - Q(s_t,\ a_t))] \quad (2)$$

### 3.3 Reward

The goal for RL is to maximise rewards through incrementally mapping states to actions. We track the achieved response time when the agent takes action at varying levels of CPU utilization and calculate a reward post-action using Algorithm 1. When $currentRT$ is a value below or equal to the $TargetRT$ and thus satisfying SLA (line 2), we assign varying reward levels. When the action is a no-action (line 3), we give the maximum reward of 1 as no migration cost was incurred and SLA is met. This helps the agent learn that no-action is rewarding for the given node state and dynamically learn the threshold to perform a migration.

When the VM was not meeting its SLA target, as in stressed (line 5), and is now meeting SLA, we want to assign a utility that reflects this as a positive action. Additionally, we want the agent to increase the utilization of target nodes by choosing a target that can host additional VMs and meet SLA. For example, when the TargetRT is 0.5 and the currentRT is 0.4, the reward will be 0.8. When currentRT is 0.3, the reward is 0.6, meaning a higher reward where VMs response time is closest to the target SLA. This has the effect of the RL agent choosing higher utilization target groups.

To learn a dynamic threshold, the RL agent will perform exploratory actions, including no-action and VM migration, using an $\epsilon$-greedy policy to Q. When the agent migrates a VM that is not stressed (line 8), we want to assign a reward that represents closeness to targetRT, with the agent receiving a higher reward when the previousRT of the VM is closest to TargetRT. As we use previousRT in the reward, this iteratively helps the agent learn the node state that maximises reward and thus a threshold for migration. For example, if the agent migrates a non-stressed VM and the previousRT is 0.4, TargetRT is 0.5, then reward is 0.9. This would be a rewarding action for the given node's state. However, a reward of 1 would have been given in a no-action. This would iteratively help the agent discover a dynamic threshold by choosing the more rewarding action.

When $currentRT$ is above the $TargetRT$ for the VM, thus causing SLA violation, we penalise the action (line 10) by using a clamp function to a maximum of $-1$. This helps the agent learn the actions that can cause SLA violations, such as migrating to a highly loaded target node or performing a no-action when the source is highly loaded. By receiving a negative reward, the agent learns the node state, thus threshold, that cause SLA violation.

Algorithm 1 helps the agent to learn actions that maximise the reward for a given node state, rewarding actions that meet SLAs, increasing CPU utilization and penalising actions that violate SLAs.

**Algorithm 1** VM Reward

```
1: procedure VMReward(VM)
2:     if currentRT ≤ TargetRT then
3:         if VM.action == NoAction then
4:             reward ← 1
5:         if VM.wasStressed() then
6:             reward ← currentRT / TargetRT
7:         else
8:             reward ← previousRT / TargetRT
9:     else
10:        reward ← clamp(TargetRT − currentRT, -1)
```

## 4 Experimental Setup and Evaluation

We use simulation to facilitate the rapid development of experiments of large data centres. We have selected DCSim [35] because of its extensibility. We utilise the hybrid hierarchical decentralized architecture from our earlier work [19], and combine it with a new dynamic threshold detection using RL.

### 4.1 Experiments

This section evaluates our proposed dynamic threshold discovery approach and its ability to improve SLA violations and node CPU utilization. We consider our proposal under varying workloads and compare our proposal to several overload detection approaches that are effective in the literature. These are Static, Interquartile Range, Median Absolute Deviation, Local Regression. Each of the dynamic threshold approaches [9] is combined with a VM and target selection policy. We additionally compare the proposed approach to our earlier work [21] (RL1). Table 1 shows how we combine these in our experiments.

For workloads, we use public traces included in DCSim: Google 1 and Google 3. Additionally, we use a mixed workload, which comprises traces from Google 1, Google 3, Clarknet and EPA, which are included traces in DCSim. For the RL parameters in Equation 2, we use $\alpha = 0.5$, and $\gamma = 0.7$ [8].

**Table 1.** Approaches used in experiments

| **Comparison Approach** | **Overload Detection** | **VM Selection** | **Target Selection** |
|---|---|---|---|
| Static | Static [9] | Highest CPU | Heuristic [9] |
| IQR | IQR [9] | | |
| MAD | MAD [9] | | |
| LR | LR [9] | | |
| RL 1 | VM Response Time [21] | Stressed VM | RL 1 |
| RL 2 | VM Response Time & Dynamic Threshold (Proposed) | Stressed VM & Highest CPU | RL 2 |

### 4.2 SLA Violations

SLA is the agreement between a CP and a customer and typically specifies a minimum quality of service threshold. In our case, this is VM response time, which we regularly collect for all VMs, and we use it to evaluate if VMs are meeting their SLA targets.

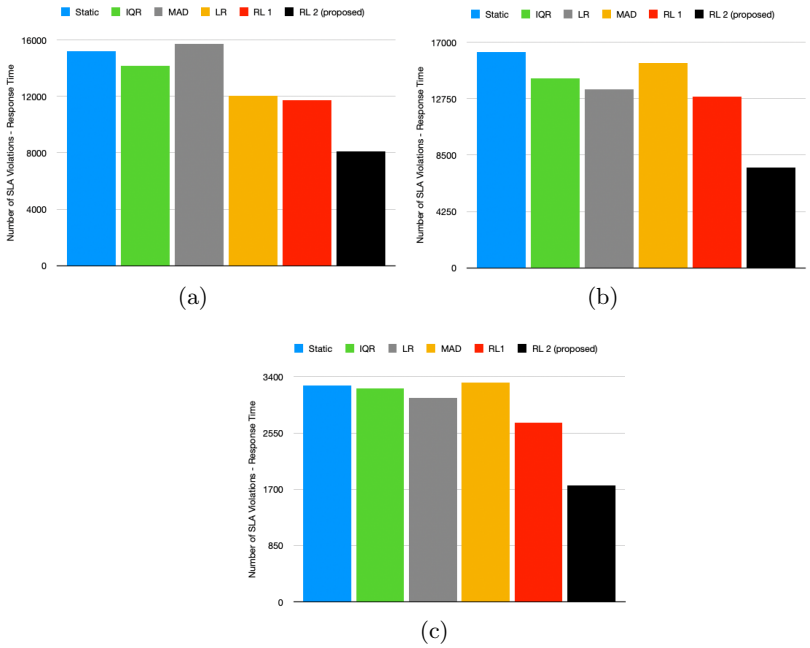



**Fig. 3.** Number of SLA violations : a) Google 1, b) Google 3 , c) Mixed workload

This experiment runs the Google 1, Google 3 and Mixed workloads [35] to evaluate how the stress detection approaches perform on SLA violations. We use an arrival rate of 90 new applications per hour for this experiment, with each application running for 10 hours before shutting down. The experiment simulates 24 hours of elapsed time, and we use 500 nodes in this experiment.

The results for Google 1, Google 3 and a mixed workload are shown in Figure 3a, 3b and 3c respectively and show RL significantly reducing the number of SLA violations for all workloads. On the Google 1 workload, RL2 achieved fewer migrations against all approaches as it incorporates the VM response time and thus directly focuses on controlling the VM performance and migrates VMs when it is close to entering a stressed state. On the Google 3 workload, RL2 achieved fewer migrations than all approaches except RL1, by 12%. On the Mixed workload, RL2 achieved fewer migrations than all approaches except RL1, where it has a comparable number of migrations. The additional migrations in RL2 are due to the exploratory discovery of the migration threshold. They occur at a low probability ($\epsilon$) and tend to be distributed throughout the lifecycle of a VM, and therefore have no impact on SLA violations.

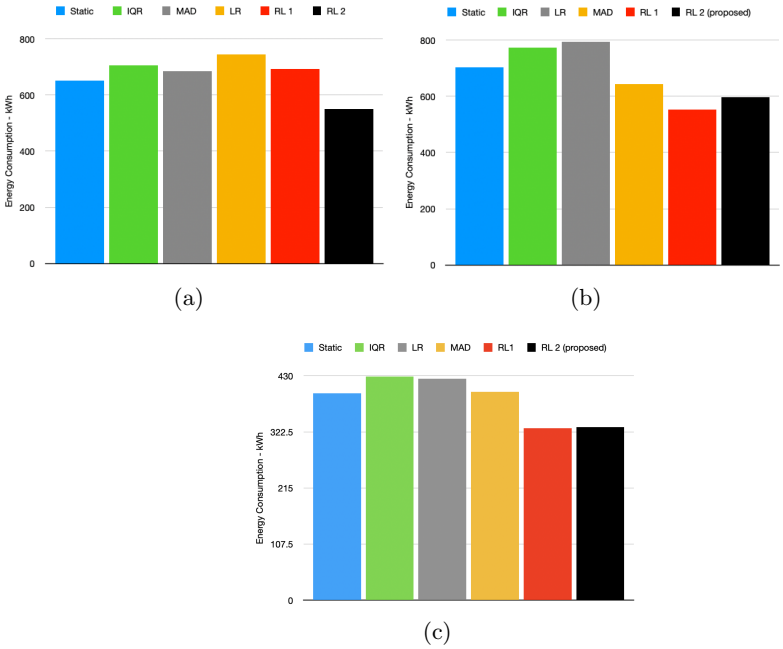



**Fig. 4.** Energy consumption (KWh) : a) Google 1, b) Google 3 , c) Mixed workload

### 4.3 Energy Consumption

The workloads and VM arrival rates, described in the previous section, create a load that requires more than 70% of the CPU resources of active nodes. DCSim can track total energy consumption within the simulated data centre by mapping CPU utilisation of a node to a defined energy consumption amount and tracking this accumulatively for all nodes. The energy consumption results for Google 1, Google 3 and Mixed workload are shown in Figure 4a, 4b and 4c respectively and show the proposed approach consistently consumed less energy compared to the dynamic threshold heuristics. On the Google 1 workload, our approach (RL2) used 20% less energy than RL1. However, RL2 used 7.9% more energy on the Google 3 workload. This difference is likely due to the number of performed migrations, where RL2 consumes more energy when it performs more migrations, which is the case on the Google 3 workload. This hypothesis is further supported by the result for the mixed workload, where RL1 and RL2 have comparable energy consumption and number of migrations. RL 2 performs some migrations to discover a dynamic threshold, and while these do converge, the discovery process will cause some migration and powering on some nodes, leading to energy consumption. The agent behaviour typically offsets this to increase the utilization of nodes and delay VM close to SLA violations, as exhibited in Google 1, where RL 2 used 27% fewer active nodes than RL1.

### 4.4 Learning Threshold Assessment

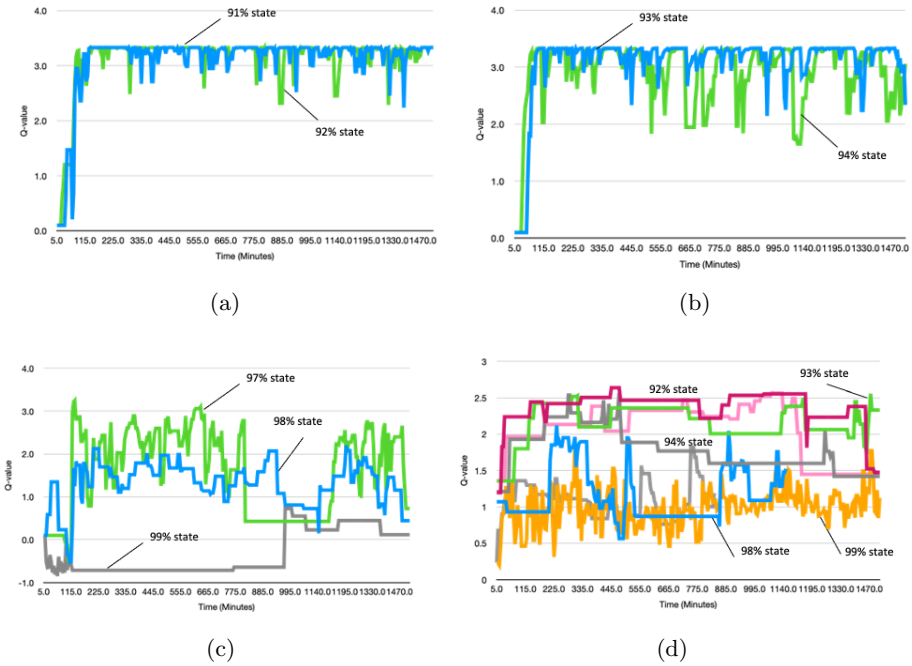



**Fig. 5.** Q-value for different actions at various CPU utilization levels: a) no-action 91% & 92%, b) no-action 93% & 94%, c) no action 97% to 99%, d) migrations at 90%+

Our approach aims to discover a dynamic threshold for migrations, which delays the migration close to the point where SLA violations would start to occur. Our RL agent aims to perform no-action on VMs up to the point they would enter a stressed state and accumulate reward as described in Section 3.3. Figure 5 shows the accumulated learning of agents during the Google 3 workload for different states and actions. The RL agent accumulates Q-value through being in a state and executing a particular action, thus a Q-value for every state-action pair at any given time. States visited less frequently will accumulate rewards slower than more frequent states. Figure 5a shows the agent reward for a no-action at CPU utilization 91% and 92%. These are typically higher than the reward the agent receives for migrating at the same CPU utilization, shown in Figure 5d, and this leads the agent to perform more no-action. As the CPU utilization increases, we examine some variability on the received reward, shown in Figure 5b. This becomes more pronounced on higher CPU utilization levels of 97% and 98% suggesting this is the threshold during this experiment, shown in

Figure 5c. At 99% CPU utilization, the agent receives a negative or low reward for a no-action and a higher reward for performing a migration, and will choose the migration at this CPU utilization.

Due to limited space on the paper, we have omitted the results for utilization 0 to 80%. These levels occur more frequently and converge rapidly within a few hours of the learning. This leads our RL agent to execute more no-action and results in fewer VM migrations than the dynamic heuristics.

## 5 Conclusion and Future Works

This paper proposes a dynamic approach to setting the CPU threshold level used to migrate VMs, using RL. Our approach can learn the migration point dynamically based on the current environment and adjust the migration point when there are changes in the managed environment. Through experimentation, we have shown that the approach can reduce SLA violations and can typically find a more optimal migration point and increase node utilization, compared to four other heuristics from the literature: Static, Interquartile Range, Median Absolute Deviation and Local Regression.

Our approach does not require a model of the environment or managed VMs, making it likely to perform well in executing a range of VMs. However, it is currently limited by using a single dynamic threshold for the entire node, irrespective of the specific behaviour of the individual VMs running on the node. A more robust approach could extend our mechanism and track VMs CPU to relate the properties of the individual VMs to the node's CPU. This could further optimise our proposed method by discovering lower CPU utilization points that reduce SLA violations and higher CPU utilization points that further increase node utilization. We aim to investigate this in future work.

## References

1. A. El-Moursy, A., Abdelsamea, A., Kamran, R., Saad, M.: Multi-dimensional regression host utilization algorithm (mdrhu) for host overload detection in cloud computing. Journal of Cloud Computing **8**(1), 8 (2019). https://doi.org/10.1186/s13677-019-0130-2, https://doi.org/10.1186/s13677-019-0130-2
2. Addis, B., Ardagna, D., Panicucci, B., Squillante, M.S., Zhang, L.: A hierarchical approach for the resource management of very large cloud platforms. IEEE Transactions on Dependable and Secure Computing **10**, 253–272 (2013)
3. Alarifi, A., Dubey, K., Amoon, M., Altameem, T., El-Samie, F.E.A., Altameem, A., Sharma, S.C., Nasr, A.A.: Energy-efficient hybrid framework for green cloud computing. IEEE Access **8**, 115356–115369 (2020). https://doi.org/10.1109/ACCESS.2020.3002184
4. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: 2012 IEEE Network Operations and Management Symposium. pp. 204–212. IEEE, Washington, DC, USA (April 2012)

5. Almeida, J., Almeida, V., Ardagna, D., Cunha, Í., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. Journal of Parallel and Distributed Computing **70**, 344–362 (Apr 2010)
6. Arabnejad, H., Pahl, C., Jamshidi, P., Estrada, G.: A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 64–73 (2017). https://doi.org/10.1109/CCGRID.2017.15
7. Bahati, R.M., Bauer, M.A.: Towards adaptive policy-based management. In: 2010 IEEE Network Operations and Management Symposium - NOMS 2010. pp. 511–518 (2010). https://doi.org/10.1109/NOMS.2010.5488472
8. Barrett, E., Howley, E., Duggan, J.: Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurrency and Computation: Practice and Experience **25**(12), 1656–1674 (2013). https://doi.org/doi.org/10.1002/cpe.2864
9. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience **24**, 1397–1420 (Sep 2012)
10. Bibal Benifa, J.V., Dejey, D.: Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment. Mobile Networks and Applications **24**(4), 1348–1363 (2019). https://doi.org/10.1007/s11036-018-0996-0
11. Bu, X., Rao, J., Xu, C.Z.: Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. IEEE Transactions on Parallel and Distributed Systems **24**(4), 681–690 (2013). https://doi.org/10.1109/TPDS.2012.174
12. Dabbagh, M., Hamdaoui, B., Guizani, M., Rayes, A.: An energy-efficient VM prediction and migration framework for overcommitted clouds. IEEE Transactions on Cloud Computing **6**(4), 955–966 (2018). https://doi.org/10.1109/TCC.2016.2564403
13. Dutreilh, X., Kirgizov, S., Melekhova, O., Malenfant, J., Rivierre, N., Truck, I.: Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow. In: 7th International Conference on Autonomic and Autonomous Systems (ICAS'2011). pp. 67–74. Venice, Italy (May 2011), https://hal-univ-paris8.archives-ouvertes.fr/hal-01122123
14. Feller, E., Rilling, L., Morin, C.: Snooze: A scalable and autonomic virtual machine management framework for private clouds. In: IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). pp. 482 – 489 (2012)
15. Garg, V., Jindal, B.: Energy efficient virtual machine migration approach with SLA conservation in cloud computing. Journal of Central South University **28**(3), 760–770 (2021). https://doi.org/10.1007/s11771-021-4643-8, https://doi.org/10.1007/s11771-021-4643-8
16. Ghahramani, M.H., Zhou, M., Hon, C.T.: Toward cloud computing qos architecture: analysis of cloud systems and cloud services. IEEE/CAA Journal of Automatica Sinica **4**(1), 6–18 (2017)
17. Ghanbari, H., Simmons, B., Litoiu, M., Barna, C., Iszlai, G.: Optimal autoscaling in a iaas cloud. In: Proceedings of the 9th International Conference on Autonomic Computing. pp. 173–178. ICAC '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2371536.2371567
18. Hu, Y., Wong, J., Iszlai, G., Litoiu, M.: Resource provisioning for cloud computing. In: Proceedings of the 2009 Conference of the Cen-

ter for Advanced Studies on Collaborative Research. pp. 101–111. CASCON '09, IBM Corp., USA (2009). https://doi.org/10.1145/1723028.1723041, https://doi.org/10.1145/1723028.1723041
19. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Shdf - a scalable hierarchical distributed framework for data centre management. In: 2017 16th International Symposium on Parallel and Distributed Computing (ISPDC). pp. 102–111. 16th International Symposium on Parallel and Distributed Computing (ISPDC) (July 2017). https://doi.org/10.1109/ISPDC.2017.15
20. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Adaptation in cloud resource configuration: a survey. Journal of Cloud Computing **5**(1), 1–16 (2016)
21. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Scalable virtual machine migration using reinforcement learning. to be published in Journal of Grid Computing (2021)
22. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation Computer Systems **26**, 871–879 (June 2011)
23. Jamshidi, P., Pahl, C., Mendonça, N.C.: Managing uncertainty in autonomic cloud elasticity controllers. IEEE Cloud Computing **3**(3), 50–60 (2016). https://doi.org/10.1109/MCC.2016.66
24. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: International Conference on Distributed Computing Systems. pp. 62–73. International Conference on Distributed Computing Systems, IEEE, Washington, DC, USA (2010)
25. Kulshrestha, S., Patel, S.: An efficient host overload detection algorithm for cloud data center based on exponential weighted moving average. International Journal of Communication Systems **34**(4), e4708 (2021). https://doi.org/https://doi.org/10.1002/dac.4708, https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4708, e4708 dac.4708
26. Minarolli, D., Mazrekaj, A., Freisleben, B.: Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing. Journal of Cloud Computing **6**(1), 4 (2017). https://doi.org/10.1186/s13677-017-0074-3, https://doi.org/10.1186/s13677-017-0074-3
27. Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M.: Efficient resource provisioning for elastic cloud services based on machine learning techniques. Journal of Cloud Computing **8**(1), 5 (2019). https://doi.org/10.1186/s13677-019-0128-9
28. Nouri, S.M.R., Li, H., Venugopal, S., Guo, W., He, M., Tian, W.: Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. Future Generation Computer Systems **94**, 765 – 780 (2019). https://doi.org/doi.org/10.1016/j.future.2018.11.049
29. Padala, P., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K.: Adaptive control of virtualized resources in utility computing environments. In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. pp. 289–302. EuroSys '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1272996.1273026, https://doi.org/10.1145/1272996.1273026
30. Quesnel, F., Lèbre, A., Südholt, M.: Cooperative and reactive scheduling in large-scale virtualized platforms with dvms. Concurrency and Computation: Practice and Experience **25**(12), 1643–1655 (2013). https://doi.org/https://doi.org/10.1002/cpe.2848, https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.2848

31. Rao, J., Bu, X., Xu, C.Z., Wang, K.: A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In: 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. pp. 45–54 (2011). https://doi.org/10.1109/MASCOTS.2011.47
32. Rao, J., Bu, X., Xu, C.Z., Wang, L., Yin, G.: Vconf: A reinforcement learning approach to virtual machines auto-configuration. In: Proceedings of the 6th International Conference on Autonomic Computing. pp. 137–146. ICAC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1555228.1555263, https://doi.org/10.1145/1555228.1555263
33. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing. pp. 5:1–5:14. SOCC '11, ACM, New York, NY, USA (2011)
34. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. Cambridge:MIT press (1998)
35. Tighe, M., Keller, G., Bauer, M., Lutfiyya, H.: Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management. pp. 385–392. Network and service management (CNSM), 2012 8th international conference and 2012 workshop on systems virtualization management (SVM) (2012)
36. Tseng, F.H., Wang, X., Chou, L.D., Chao, H.C., Leung, V.C.M.: Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm. IEEE Systems Journal **12**(2), 1688–1699 (2018). https://doi.org/10.1109/JSYST.2017.2722476
37. Wang, J.V., Ganganath, N., Cheng, C.T., Tse, C.K.: Bio-inspired heuristics for VM consolidation in cloud data centers. IEEE Systems Journal **14**(1), 152–163 (2020). https://doi.org/10.1109/JSYST.2019.2900671
38. Watkins, C.J.C.H.: Learning from Delayed Rewards. Ph.D. thesis (1989)
39. Yadav, R., Zhang, W., Li, K., Liu, C., Shafiq, M., Karn, N.K.: An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center. Wireless Networks **26**(3), 1905–1919 (2020). https://doi.org/10.1007/s11276-018-1874-1

# Chapter 5

# Conclusion

This chapter provides a critical analysis of the related work, concludes the thesis and discusses future directions. Section 5.1 provides a critical analysis of the related work in the papers contained in Chapter 4. Each of those papers has a related work section, and this section acts as an addition to place the results as a whole within their wider context. We cover each contributed paper and critically analyse papers from the closely related literature. Section 5.2 summarises the thesis and relates the outcomes to the research objectives, described in Section 1.2. The chapter concludes by discussing future research directions in Section 5.4.

## 5.1 Critical Analysis of Related Work

### 5.1.1 Contribution 1: Adaptation in Cloud Resource Configuration: A Survey

This paper surveys resource reconfiguration, covering 40+ publications that focus on the adaptation of computing resources in a cloud context. The chosen publications appeared in cloud focused journals and conferences. The survey contributions are a definition for cloud adaptation and a classification that we use to survey the literature.

Multiple surveys pull together different features of cloud resource management [51, 33, 47, 148]. Some of the surveys focus on the cloud provider (CP) perspective, and others on the cloud customer perspective.

In [51] the authors survey elastic approaches from a cloud customer perspective, providing a high-level overview of the methods, and identified several cloud customer open challenges, including lack of standardised APIs and startup time of requested

resources. Our survey is different and focuses on the cloud provider perspective, emphasising adaptation. Another survey focusing on the cloud customer perspective is the work in [33], where the authors survey autoscaling and classify the literature based on the adaptation techniques used. Our survey focuses on the CPs persecutive and thus included VM migration and server consolidation; in contrast to the authors, only migration was covered. Additionally, our survey covers a more extensive taxonomy of cloud resource management, including the adapted resource, adaptation objective, and decision making engine architecture used for adaptation.

Examining surveys that focus on the CP perspective. In [98], the authors survey the VM allocation problem, models and algorithmic approaches. The survey covered the literature that examines single and multiple data centres and argued that methods are primarily disjoint, with a limited number addressing a combination of the two. In contrast, our survey covered a more extensive taxonomy of adapted cloud resources and covers the architecture of the decision making engine used for adaptation. Additionally, we highlight the scalability challenges in non-decentralized approaches. In [26], the authors comprehensively discuss strategies to efficient data centres, choosing to focus on energy consumption. Our survey covers energy consumption as an adaptation objective, SLA and revenue objectives. The survey [47] is closely related to our work and similarly examined the techniques used for resource allocation, including heuristics, control theory, queuing theory and machine learning, concluding that heuristics are the most prevalent in the literature. The authors also concluded that CPU and memory are the most monitored resources. In comparison, our survey has more coverage of architectures used in decision making, their properties and the impact of the architecture on cloud management scalability.

All the related surveys chose a different classification scheme to our work. In contrast, our work focuses on adapting resource configuration and analysing factors that influence adaptation. Additionally, we investigate factors affecting the scalability of the various proposals in the literature. While there is some overlap, no other survey uses our chosen dimensions.

### 5.1.2 Contribution 2: SHDF - Scalable Hierarchical Distributed Framework for Data Centre Management

This paper proposes a hybrid decentralized hierarchical Management Framework (MF) to manage public cloud resources, achieve high scalability and meet SLAs. In the

following section, we examine the properties of our proposal against decentralized and hierarchical MFs from the literature.

The proposal in [152] adopted a decentralized design where a node is autonomous and is responsible for resource allocation. Each node dynamically adapts existing placements in response to a change in workload needs, and the approach can manage an extensive infrastructure with thousands of nodes. Similarly, the proposal in [114] presents a decentralized approach that aims to be scalable and energy-efficient in its management of VMs provisioned in enterprise clouds. A core concept in the approach is that computation resources are organised into an overlay network called a hypercube. The hypercube automatically scales up and down as resources are changed in response to workload changes. Each node in the approach operates autonomously and manages its workload. Overutilized nodes attempt to migrate their VMs to their neighbours in the hypercube to avoid SLA violations. Underutilized nodes aim to shift their workload to their neighbours and switch off. Similarly, in our approach, each node has a component to decide the workloads that run on the node and an overlay manager that maintains information relating to the overlay, enabling a node to participate and interact with a designated set of neighbours. Members of the overlay regularly interact by selecting a random neighbour to exchange metric information. In contrast, our approach operates in a hybrid decentralised hierarchical manner, with higher-level controllers assisting the lower decentralised controllers. This enables our approach to perform consolidation at the cluster level, whereas other decentralized approaches perform load balancing between two nodes. Additionally, our approach can utilise the collected metrics to reach out to a more significant portion of an overlay during a stressed VM state, as opposed to the author's approach, a node only operates with a randomly selected node, which can result in VMs being in a stressed state for some time, leading to more SLA violations. Additionally, in our approach, a stressed node can escalate outside the overlay if there is no suitable node to migrate a VM to.

Traditional hierarchical MFs [10, 79, 13] utilise a controller running in a centralized manner within the scope of a cluster or rack. In Mistral [79], multi-level hierarchical controllers manage different subsets of nodes. The lowest level controllers manage a smaller number of nodes at the rack level, and at the next cluster level, a controller manages nodes owned by multiple lower-level controllers. At the rack level, the approach operates in a centralized manner, with nodes sending metric information to and receiving migration instructions from the rack controller. Our proposed hybrid approach similarly has a cluster level controller that connects to lower-level nodes

and enables a large scope for consolidation. In contrast, our lowest level controller manages a single node and allows it to be completely autonomous and cooperate with other autonomous nodes to facilitate VM migration. Nodes can receive migration requests from neighbouring nodes or higher-level controllers, and each node can choose to accept or reject these requests. This results in the cluster controllers performing less computation than the traditional hierarchical approach, resulting in higher system scalability.

A hierarchical architecture is used in [10]. At the highest level of the hierarchy, a central manager focuses on a timescale of 1 day and divides the infrastructures into clusters to reduce the need for fine-grain adjustments by lower-level controllers. At a lower hierarchy level, application managers centrally operate each cluster and act on a 1-hour scale, focusing on VM placement and capacity allocation to VMs running on the same node. Additionally, application managers perform cheaper operations such as capacity allocation and frequency scaling more frequently, every 5-15 minutes. The hierarchical architecture reduces the monitoring data required in the approach, with application managers providing an aggregate of metrics. Experiments show that application managers can perform their periodic operations and solve for 1200 nodes in less than 1 minute. The approach has several similarities to our proposal, with multiple controllers running at different timescales and dividing the data centre infractions into multiple clusters. In contrast, our lowest level controllers are autonomous and manage a single node and thus can operate at low timescales, 2 minutes by default, and has been tested with 50,000 nodes. Additionally, our autonomous node controllers can escalate a stressed VM outside of its original cluster on-demand, thus speeding up the time to resolve a stressed state versus the approach in [10].

### 5.1.3 Contribution 3: A Hierarchical Decentralized Architecture to enable Adaptive Scalable Virtual Machine Migration

Scalability of a system is defined as the ability to meet additional workload requirements by adding a proportional amount of resources and maintaining its performance [71]. In this paper, we extend a preliminary version of our work [66], by implementing and evaluating QoS metrics on multiple policies from the literature. Our goal is to demonstrate improved QoS metrics compared to centralized, hierarchical and decentralized architectures for a variety of MAs. By achieving improved QoS performance metrics, our proposal should scale better than the examined architectures; the extent to

which this is the case is explored through empirical evaluation.

The authors in [99] propose a test environment to evaluate the effectiveness of different VM placement algorithms running within a centralized Management Framework (MF) by extending the existing simulation environment of cloudSim [36]. The authors undertake an empirical comparison of 7 management algorithms (MAs) and propose a methodology to evaluate the MAs. Experiments concluded that a cluster of MAs performed well on some examined metrics. We follow a similar approach of extending an existing simulation environment, integrating 8 MAs from the literature, and evaluating their properties. In contrast, our approach additionally includes a comparison with multiple MFs: hierarchical, decentralized and hybrid. Thus we have a more comprehensive evaluation, and our approach also evaluated the scalability of these MFs.

### 5.1.4 Contribution 4: Scalable Virtual Machine Migration using Reinforcement Learning

This paper proposes a reinforcement learning (RL) MA that can run decentralized and achieve fast convergence towards efficient resource allocation, resulting in lower SLA violations than centralized architectures. The aim is to investigate and address some common challenges in applying RL, such as slow learning and state/action management. We also demonstrate a unique, multi-level reinforcement learning cooperation that further reduces SLA violations. We use simulation to evaluate and present our proposal in practice and compare the results obtained with an established heuristic, demonstrating significant improvement to SLA violations and higher scalability.

The authors in [22] use a Q-learning approach where agents operate in a decentralized manner, communicate directly with all other agents and autonomously make resource management decisions. To address the convergence challenge in RL, the approach parallelises agents' learning. Each agent learns the value of states it visits and shares this with neighbouring agents to learn about unvisited states. Each agent maintains a local Q table representing its learning and a Q-value representing the global learning. This is calculated by aggregating the weighted sum of Q-value estimates of all the neighbouring agents. Each agent makes decisions based on the weighted aggregation of the local and global estimates, which are typically greedy towards the current learnt Q-value. Sometimes, the agent will choose to act randomly to balance

exploration and exploitation. Similarly, our proposed RL based MA has agents running in a decentralized manner, with agents communicating directly with other agents and autonomously making resource management decisions. Additionally, we have a similar approach to parallel learning, where each agent shares its experience with neighbouring agents. In contrast, our approach has more efficient agent communication where agents cooperate with other agents in their overlay, typically placed in physical proximity, resulting in a message exchange with a smaller group of nodes and a shorter distance for messages to travel within the data centre. Additionally, our proposed approach enables decentralized nodes to escalate migration requests to other nodes outside their overlay and learn when to escalate.

Reinforcement learning techniques can suffer from challenges where the state and action space grow exponentially. This increases the time needed for the RL agent to explore a given environment and the space complexity in memory consumption. The approach in [35] used a heuristic method to reduce the state space to a smaller set by dividing the original state space into multiple exclusive subsets, where a range of states can fit into the same subgroup, thus reducing the state space to aid RL convergence speed. Our proposed RL MA uses a similar approach and aggregates VM's to two states: Normal and Stressed by using *Response time* as a measure for application performance [53]. To account for variation in response time during the lifetime of an application, we apply linear regression on collected response time during each monitoring epoch, which by default is every two minutes. A VM is stressed when the 95th percentile of response time during a monitoring period is above a defined SLA threshold, which by default is 500ms. We categorise the state of VMs as *Normal* when the 95th percentile of the response time is below the defined SLA level.

Our paper is similar to [112], which proposes a Q-learning controller and decentralized approach where each node is responsible for maintaining its SLA performance. The approach can add nodes and scale down by shutting down excess nodes to save energy consumption. To combat the state space challenge in Q-learning, the approach uses a reduced state space consisting of two types: system and applications state. System state reflects the level of utilization of resources of a node, and applications state represents the performance of each application running on the node. System and applications states are classified into a few categories: normal, warning, and critical. For the RL actions, the authors use a policy to reduce the number of actions for each state, with actions categorised into two groups of scale-down and scale-up. When the system is not meeting its defined goal, scale-up actions are taken; scale-down actions are

taken when the system is in normal condition. The RL agents share their state/action through a knowledge base to speed up learning. When an agent encounters a new state, it checks the knowledge base for known actions. Similarly, our proposed RL MA runs in a decentralized architecture, uses state and action reduction to address challenges with Q-learning, and our agents share knowledge to speed up learning. In contrast, our approach selects a granular action to a set of target nodes, enabling RL to learn a more precise set of actions for a given state. When migrating a VM, our RL agent can identify a group of target nodes based on their CPU utilisation. By default, we use 10 groups, 10%, which creates target groups from 0 to 9. For example, action group1 means to migrate the stressed VM to a node with an average CPU utilisation of 10% to 19%. Once an action is selected, we use a greedy policy to select the first available node that fits the action group. Additionally, our approach utilises the features of the hybrid MF from our earlier work and can learn when to escalate a migration outside of the overlay, thus speeding up the resolution of a stressed VM state.

While there have been attempts at examining the scalability of approaches based on RL [35, 120], these tend to be at a small scale that is not representative of the size of the infrastructure in public clouds. We propose a highly scalable RL approach and examine its ability to manage an extensive infrastructure with thousands of nodes.

### 5.1.5 Contribution 5: Utilization Efficient VM Migration using Reinforcement Learning

This paper presents an approach to dynamically set the CPU utilization level for VM migration. It monitors VM response time and discovers a CPU threshold where response time would increase beyond a target SLA level. This threshold is then used for VM migration. The approach uses RL to learn when it is rewarding to migrate a VM. The reward function aims to increase CPU utilisation and attaches a penalty to overachieving SLAs. We evaluate the approach against four dynamic heuristics from the literature, and the results show a significant reduction in SLA violations and an increase in CPU utilization of active nodes.

Reactive approaches are typically implemented using threshold techniques [41] by triggering adaptation when a node's utilization reaches a given level. Several heuristics have been proposed in the literature, including [25, 155, 102, 86, 5].

The heuristic in [25] calculates an upper threshold through statistical analysis of historical CPU utilisation. Some of the approaches incorporate multiple node metrics

by creating a composite metric for overload detection, that combines CPU utilization, memory and network BW utilization [5]. However, these approaches do not incorporate VM performance such as response time in setting the node CPU threshold.

Approaches that did consider VM performance include [121]. They propose to manage the resources assigned to a VM and monitor performance feedback from each VM. The approach optimises VM performance by learning the resource allocation that enhances response time and throughput. The reconfiguration process happens periodically on a predefined time interval. A controller fetches the current state of VMs and computes valid adaptation actions that can adjust CPU and memory assigned to a VM. The approach has similarities to our proposal. It uses RL to discover a rewarding configuration that improves VM performance. In contrast, the approach focuses on parameters for vertically scaling VMs and does not cater for VM migration.

The approach in [35] proposes CoTuner, for coordinated configuration of VM resources and parameters of their applications. Each VM has an agent that monitors the VM and adapts its configuration to the environment. Reconfiguration actions take place periodically at predefined time intervals. The RL method receives performance feedback and updates the VM and application configuration. CoTuner can adjust both CPU and memory VM assignments. For applications, CoTuner can change parameter settings. The approach has similarities to our proposed approach. It uses RL to discover a configuration that improves VM performance. In contrast, the approach focuses on parameters for application configuration and does not cater for VM migration.

## 5.2 Conclusion

Adaptation of cloud resource configuration is applied to remap resource assignments, and the majority of MFs used in literature are centralized with some that are hierarchical or decentralized [90]. Centralized MFs use an engine with a global control of the entire infrastructure that performs all the phases required for the resource mapping, including monitoring, calculating the schedule, and applying the resource mapping. Hierarchical MFs typically divide the infrastructure into multiple sections with a decision engine in each section. While this improves scalability compared to centralized MFs, as the size of the managed infrastructure increases, the approach suffers similar challenges. Decentralized MFs distribute the management of the infrastructure without a centralized controller. However, they can be limited by the partial view of the

managed infrastructure. Research *Obj1* is to characterise the attributes of cloud management systems, and *contribution 1* has shown all of these architectures have their merits. Centralized has a global view and a simple design but requires explicit measures to attain reliability. Hierarchical architectures have increased scalability compared to centralized and also require measures to achieve reliability. Decentralized architectures have no central controller and have been shown to scale to manage a large number of nodes and typically have reliability built in the design.

This thesis is grounded on the hypothesis that solving the scalability challenge in mapping workloads to resources starts with tackling scalability in the MF. In addressing *Obj2*, we propose a scalable hierarchical decentralized framework that combines the advantages of both approaches and mitigates their limitations, leading to *contribution 2*. The approach consists of hierarchical controllers operating at different scopes. On the lowest level, every node in the infrastructure is autonomous, manages its resource allocation and cooperates with other peer nodes. When a VM is in an SLA violation state, a node cooperates with local nodes and can escalate through the hierarchy to find a target for the stressed VM. For consolidation objectives, a cluster-level controller can view all nodes in a cluster and include more nodes in the optimization. The proposed MF retains the advantages of decentralized architectures and mitigates their disadvantages by enabling a cluster level view during consolidation. Weaknesses of the hierarchical architecture are mitigated through the decentralized approach that allows nodes to cooperate with other nodes during VM migration, only resorting to the cluster level controller in an escalation case. We have validated this approach using simulation and scaled our experiments to 50,000 nodes.

To further evaluate the proposed hybrid MF, addressing *Obj3* develops an environment to assess and compare the performance of multiple management algorithms (MAs) from the literature within a simulation toolkit. We additionally compare the performance of the MAs with hierarchical, decentralized and centralized MFs, leading to *contribution 3*. All MAs retained their SLA performance properties when running on the hybrid MF. Some exhibited higher SLA performance due to a reduced search space and autonomous properties of the Hybrid MF. The improvements included fewer SLA violations, lower network traffic utilisation, and improved scalability to manage resources as the number of nodes in the data centre increased compared to centralized, hierarchical, and decentralized architectures. Additionally, this evaluation demonstrated the feasibility of separating the MF and MA, and the ability to integrate the hybrid MF with other MAs to investigate cloud resource management.

The MA is the decision making component that assigns workloads to infrastructure resources by regularly assessing the satisfaction of such assignments in achieving a given objective, which typically includes SLAs. A key aspect of MAs is the technique used to perform the decision making process. While heuristics are common in the literature, cloud workloads are heterogeneous, with different resource requirements. Heuristics typically require predefined knowledge of the problem they solve, reducing their applicability. While machine learning models have shown to be effective in their application in cloud resource management, a limitation of these approaches is that they require offline training and thus require retraining to take advantage of new data. In contrast, RL can operate online even when a complete environmental model is unavailable, a valuable property in cloud environments. In addressing *Obj4*, we hypothesise that the proposed hybrid MF can be paired with a MA to utilise its features and lower SLA violations further. We investigate and develop an RL-based MA that reduces the state and action space and uses a unique multi-level RL agent cooperation between different levels of the hybrid hierarchy. Our approach integrates well into the hybrid MF, leading to *contribution 4*. We evaluate its performance using simulation and compare the results with an established heuristic, demonstrating significant improvement to SLA violations and high scalability.

Our research has shown that many of the approaches in the literature use a reactive, threshold-based approach to detect node overload. A key element to the threshold-based approach is the assumption there is a high chance that an overload occurs when a node's utilization exceeds the set threshold. Thus, the threshold level creates an association between a node metric, e.g. CPU utilization, and SLA violation. However, the CPU utilization where SLA violations occur is based on the application and the node configuration. Creating a single threshold for all applications and node configurations is incredibly difficult. While the current approaches can reduce overload, they can limit the utilization gains that can be achieved as they leave unused slack for each node. Additionally, threshold approaches can trigger unnecessary migrations as exceeding the set threshold does not necessarily equate to an SLA violation [41]. Given these challenges, addressing *Obj5* proposes a dynamic approach to set CPU threshold for VM migration. We perform online learning and incorporate VM performance in setting a CPU utilization threshold to use for migration. This enables the threshold to factor in the dynamic nature of cloud environments and their heterogeneous workloads, leading to *contribution 5*. To discover a dynamic threshold, each node periodically performs experimental VM migrations at varying CPU utilisation levels. RL calculates a reward

for each used threshold level and converges on a migration threshold. We show the approach can reduce SLA violations and can find a more optimal migration point and increase node utilization, compared to four other heuristics from the literature.

## 5.3 Choosing a Simulation Toolkit

Experimenting with MAs and MFs at the data centre scale is impractical and can be very costly. Thus, there is a need for simulation tools to allow rapid development and evaluation of data centre management approaches. There are several simulation tools available [162, 46]. We have chosen to use DCSim, an extensible simulation framework for simulating a data centre hosting an Infrastructure as a Service cloud. DCSim allows a VM to use more CPU than reserved, up to an amount that does not impact other VMs. This means CPU can be overcommitted until it affects other VMs. In DCSim, an application is modelled as an interactive multi-tiered web application. Each application has: (i) think time, which is the period it takes for a simulated user to act between requests, (ii) a workload component and (iii) a request service time, which is the amount of time required to process each incoming request. The workload defines the current number of clients connected to the application, which can change at discrete points in the simulation based on a trace file. The resource requirements are defined as its resource size, which is the expected amount of CPU, memory, bandwidth and storage. DCSim treats bandwidth and storage as fixed requirements. However, CPU requirements can be varied across the simulation based on the VM demands. DCSim applies a cost to VM migration, including the time taken for migration, as a function of memory consumed by a VM, and factors in the bandwidth required for the VM migration on the hosting node. Additionally, the DCSim simulation environment has a more accurate representation of metric propagation, compared to cloudSim [36], which assumes the utilization metrics of all nodes can be read directly instead of being sent as messages, as in DCSim.

## 5.4 Future Directions

The remaining section of this chapter introduces some potential research directions that can extend the work conducted in this thesis.

### 5.4.1 Adaptive parameter selection

The hybrid MF and RL MA use multiple parameters, and there is an opportunity for these to be dynamically adjusted based on the current cloud conditions to optimise performance further.

- **Periodic node state checks**. Each autonomous node in the hybrid MF needs to check its state and VMs running on it regularly. By default, this is every 2-minutes. However, there is an opportunity to set this value by learning it dynamically. This could enable node checks to be performed more frequently when undergoing volatility and less regularly during more stable periods.

- **Periodic node state exchanges**. Each autonomous node in the hybrid MF exchanges its state with a randomly selected node within the same overlay, which helps propagate node metrics. By default, this is every 1-minute. However, there is an opportunity to set this value by learning it dynamically. This could enable node checks to be performed more frequently when an overlay is undergoing higher VM placement or VM migration levels, less regularly during more stable periods.

### 5.4.2 Initial VM Placement using RL

The objective of a VM Placement is to map customer workloads onto Cloud Providers resources in a way that achieves a particular objective, such as reducing energy consumption or ensuring SLAs are met. VM placement consists of two parts: *initial* VM placement, which refers to the first allocation of each VM to a node in the data centre, and VM *migration* or relocation, which involves the revision of an earlier placement decision.

We have focused on VM *migration* in this thesis, and both the hybrid MF and RL MAs that we have developed can be extended to cater for the initial VM placement, where RL can be used to decide on target nodes for initial placement. The reward outcomes from the initial placement can help the RL agent make migration decisions, which could further speed up the convergence time of the RL approach.

### 5.4.3 VM consolidation using RL

Conserving energy consumption is widely researched in the literature. However, there are still challenges with identifying the benefit versus migration cost during consolidation. We recognise this as additional research that could be improved with an RL MA that can learn a policy from the current environmental conditions and decide on the benefits of applying consolidation. While some of the literature already investigated using RL in consolidation [48, 38, 128], there is an opportunity for a new MA to learn how to select a VM for consolidation and target node(s) to consolidate VMs onto.

### 5.4.4 Containerisation

Our research on this thesis project has shown that cloud resource management primarily focuses on VMs. However, is there is a growing number of proposals investigating resource management of containers [55, 93]. It is feasible to extend the methods proposed in this thesis and apply them to containers. As there are likely to be more containers than VMs, we hypothesise the hybrid MF will provide even more value in managing what would be a more significant number of cloud resources. The proposed RL approach can be extended to capture container metrics, create dynamic threshold levels for container migrations, and manage efficient container migration targets.

### 5.4.5 Additional RL state

The representation of the state is key to the RL decision making process. To overcome the state space dimensionality challenge with RL, we use a reduction approach and aggregate VMs to two states: Normal and Stressed. However, there is an opportunity to expand the captured state to learn different patterns about the workload, such as time of day, which may affect workload behaviour. Additional states that could be captured include overlay and cluster metrics, enabling an RL MA to directly escalate to another part of the infrastructure for a given infrastructure state.

### 5.4.6 Real Cloud experiments

Most of the literature, including this thesis, uses simulation toolkits due to the impracticality of performing extensive large scale experiments on actual cloud infrastructure. However, running experiments on real infrastructure may add valuable insight to measure the effectiveness of many of the ideas in this thesis.

Additionally, actual infrastructure experiments will enable observation of events and interactions that may not have been modelled in the simulation environment and thus further enhance the understanding of cloud resource management.

# Bibliography


[1] Linux kvm, Accessed 2021-08-02.

[2] Vmware esxi, Accessed 2021-10-02.

[3] Xen hypervisor, Accessed 2021-10-21.

[4] University of Manchester Journal Format, December Accessed 2021-11-01.

[5] A. EL-MOURSY, A., ABDELSAMEA, A., KAMRAN, R., AND SAAD, M. Multi-dimensional regression host utilization algorithm (mdrhu) for host overload detection in cloud computing. *Journal of Cloud Computing 8*, 1 (2019), 8.

[6] ABDEL-BASSET, M., ABDLE-FATAH, L., AND SANGAIAH, A. K. An improved lévy based whale optimization algorithm for bandwidth-efficient virtual machine placement in cloud computing environment. *Cluster Computing 22*, 4 (2019), 8319–8334.

[7] ABDUL RAHMAN HUMMAIDA, NORMAN W PATON, R. S. Dynamic threshold setting for vm migration. In submission to ESOCC 2022.

[8] ABENI, L., AND FAGGIOLI, D. Using Xen and KVM as real-time hypervisors. *Journal of Systems Architecture 106* (2020), 101709.

[9] ABOHAMAMA, A., AND HAMOUDA, E. A hybrid energy–aware virtual machine placement algorithm for cloud environments. *Expert Systems with Applications 150* (2020), 113306.

[10] ADDIS, B., ARDAGNA, D., PANICUCCI, B., SQUILLANTE, M. S., AND ZHANG, L. A hierarchical approach for the resource management of very large cloud platforms. *IEEE Transactions on Dependable and Secure Computing 10* (2013), 253–272.

[11] AL-DHURAIBI, Y., PARAISO, F., DJARALLAH, N., AND MERLE, P. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing 11*, 2 (2018), 430–447.

[12] ALI-ELDIN, A., TORDSSON, J., AND ELMROTH, E. An adaptive hybrid elasticity controller for cloud infrastructures. In *2012 IEEE Network Operations and Management Symposium* (Washington, DC, USA, April 2012), IEEE, pp. 204–212.

[13] ALMEIDA, J., ALMEIDA, V., ARDAGNA, D., CUNHA, Í., FRANCALANCI, C., AND TRUBIAN, M. Joint admission control and resource allocation in virtualized servers. *Journal of Parallel and Distributed Computing 70* (Apr 2010), 344–362.

[14] ALRESHEEDI, S. S., LU, S., ABD ELAZIZ, M., AND EWEES, A. A. Improved multiobjective salp swarm optimization for virtual machine placement in cloud computing. *Human-centric Computing and Information Sciences 9*, 1 (2019), 15.

[15] ARABNEJAD, H., PAHL, C., JAMSHIDI, P., AND ESTRADA, G. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2017), pp. 64–73.

[16] ARIANYAN, E., TAHERI, H., AND KHOSHDEL, V. Novel fuzzy multi objective dvfs-aware consolidation heuristics for energy and SLA efficient resource management in cloud data centers. *Journal of Network and Computer Applications 78* (2017), 43–61.

[17] ASKARIZADE HAGHIGHI, M., MAEEN, M., AND HAGHPARAST, M. An energy-efficient dynamic resource management approach based on clustering and meta-heuristic algorithms in cloud computing IaaS platforms. *Wireless Personal Communications 104*, 4 (2019), 1367–1391.

[18] AZIZI, S., SHOJAFAR, M., ABAWAJY, J., AND BUYYA, R. Grvmp: A greedy randomized algorithm for virtual machine placement in cloud data centers. *IEEE Systems Journal 15*, 2 (2021), 2571–2582.

[19] BACHIEGA, N. G., SOUZA, P. S. L., BRUSCHI, S. M., AND DE SOUZA, S. D. R. S. Container-based performance evaluation: A survey and challenges. In *2018 IEEE International Conference on Cloud Engineering (IC2E)* (2018), pp. 398–403.

[20] BALA, A., AND CHANA, I. Prediction-based proactive load balancing approach through vm migration. *Engineering with Computers 32*, 4 (Oct. 2016), 581–592.

[21] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, Association for Computing Machinery, pp. 164–177.

[22] BARRETT, E., HOWLEY, E., AND DUGGAN, J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience 25*, 12 (2013), 1656–1674.

[23] BASET, S. A. Cloud SLAs: Present and future. *Special Interest Group on Operating Systems 46*, 2 (July 2012), 57–66.

[24] BELOGLAZOV, A., ABAWAJYB, J., AND BUYYA, R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems 28* (May 2012), 755–768.

[25] BELOGLAZOV, A., AND BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience 24* (Sep 2012), 1397–1420.

[26] BELOGLAZOV, A., BUYYA, R., LEE, Y. C., AND ZOMAYA, A. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers 82* (2011), 47–111.

[27] BERRAL, J. L., GAVALDA, R., AND TORRES, J. Adaptive scheduling on power-aware managed data-centers using machine learning. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing* (USA, 2011), GRID '11, IEEE Computer Society, pp. 66–73.

[28] BERRAL, J. L., GOIRI, I. N., NOU, R., JULIÀ, F., GUITART, J., GAVALDÀ, R., AND TORRES, J. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking* (New York, NY, USA, 2010), e-Energy '10, ACM, pp. 215–224.

[29] BIBAL BENIFA, J. V., AND DEJEY, D. Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment. *Mobile Networks and Applications 24*, 4 (2019), 1348–1363.

[30] BITSAKOS, C., KONSTANTINOU, I., AND KOZIRIS, N. Derp: A deep reinforcement learning cloud system for elastic resource provisioning. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (2018), pp. 21–29.

[31] BODÍK, P., GRIFFITH, R., SUTTON, C., FOX, A., JORDAN, M., AND PATTERSON, D. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2009), HotCloud'09, USENIX Association.

[32] BORGETTO, D., DA COSTA, G., PIERSON, J.-M., AND SAYAH, A. Energy-aware resource allocation. In *2009 10th IEEE/ACM International Conference on Grid Computing* (2009), pp. 183–188.

[33] BOTRAN, T. L., MIGUEL-ALONSO, J., AND LOZANO, J. A. Auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing 12*, 4 (Dec 2014), 559–592.

[34] BU, X., RAO, J., AND XU, C.-Z. Model-free learning approach for coordinated configuration of virtual machines and appliances. In *19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems* (Washington, DC, USA, 2011), IEEE, pp. 12–21.

[35] BU, X., RAO, J., AND XU, C.-Z. Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. *IEEE Transactions on Parallel and Distributed Systems 24*, 4 (2013), 681–690.

[36] CALHEIROS, R. N., RANJAN, R., BELOGLAZOV, A., DE ROSE, C. A. F., AND BUYYA, R. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Practice & Experience 41*, 1 (Jan. 2011), 23–50.

[37] CHAURASIA, N., KUMAR, M., CHAUDHRY, R., AND VERMA, O. P. Comprehensive survey on energy-aware server consolidation techniques in cloud computing. *The Journal of Supercomputing 77*, 10 (2021), 11682–11737.

[38] CHOU, Q., FAN, W., AND ZHANG, J. A reinforcement learning model for virtual machines consolidation in cloud data center. In *2021 6th International Conference on Automation, Control and Robotics Engineering (CACRE)* (2021), pp. 16–21.

[39] CITRIX. Xen. `http://www.xenserver.org`, 2016.

[40] COSTACHE, S., DIB, D., PARLAVANTZAS, N., AND MORIN, C. Resource management in cloud platform as a service systems: Analysis and opportunities. *Journal of Systems and Software 132* (2017), 98–118.

[41] DABBAGH, M., HAMDAOUI, B., GUIZANI, M., AND RAYES, A. An energy-efficient VM prediction and migration framework for overcommitted clouds. *IEEE Transactions on Cloud Computing 6*, 4 (2018), 955–966.

[42] DEL MESTRE MARTINS, A. L., DA SILVA, A. H. L., RAHMANI, A. M., DUTT, N., AND MORAES, F. G. Hierarchical adaptive multi-objective resource management for many-core systems. *Journal of Systems Architecture 97* (2019), 416–427.

[43] DONG, D., STACK, P., XIONG, H., AND P. MORRISON, J. Managing and unifying heterogeneous resources in cloud environments. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science* (Setubal, PRT, 2017), CLOSER 2017, SCITEPRESS - Science and Technology Publications, Lda, pp. 143–150.

[44] DONYAGARD VAHED, N., GHOBAEI-ARANI, M., AND SOURI, A. Multi-objective virtual machine placement mechanisms using nature-inspired metaheuristic algorithms in cloud environments: A comprehensive review. *International Journal of Communication Systems 32*, 14 (2019), e4068. e4068 IJCS-19-0062.R1.

[45] ENDO, P. T., RODRIGUES, M., GONÇALVES, G. E., KELNER, J., SADOK, D. H., AND CURESCU, C. High availability in clouds: systematic review and research challenges. *Journal of Cloud Computing 5*, 1 (2016), 16.

[46] FAKHFAKH, F., KACEM, H. H., AND KACEM, A. H. Simulation tools for cloud computing: A survey and comparative study. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)* (2017), pp. 221–226.

[47] FANIYI, F., AND BAHSOON, R. A systematic review of service level management in the cloud. *ACM Computing Surveys 48*, 3 (Dec. 2015), 43:1–43:27.

[48] FARAHNAKIAN, F., LILJEBERG, P., AND PLOSILA, J. Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (Feb 2014), pp. 500–507.

[49] FELLER, E., RILLING, L., AND MORIN, C. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2012), pp. 482 – 489.

[50] GAHLAWAT, M., AND SHARMA, P. Survey of virtual machine placement in federated clouds. In *2014 IEEE International Advance Computing Conference (IACC)* (2014), pp. 735–738.

[51] GALANTE, G., AND BONA, L. C. E. D. A survey on cloud computing elasticity. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing* (Washington, DC, USA, 2012), UCC '12, IEEE Computer Society, pp. 263–270.

[52] GARCÍA-VALLS, M., CUCINOTTA, T., AND LU, C. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture 60*, 9 (2014), 726–740.

[53] GHANBARI, H., SIMMONS, B., LITOIU, M., BARNA, C., AND ISZLAI, G. Optimal autoscaling in a iaas cloud. In *Proceedings of the 9th International Conference on Autonomic Computing* (New York, NY, USA, 2012), ICAC '12, Association for Computing Machinery, pp. 173–178.

[54] GHOBAEI-ARANI, M., JABBEHDARI, S., AND POURMINA, M. A. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems 78* (2018), 191 – 210.

[55] GHOLIPOUR, N., ARIANYAN, E., AND BUYYA, R. A novel energy-aware resource management technique using joint vm and container consolidation approach for green computing in cloud data centers. *Simulation Modelling Practice and Theory 104* (2020), 102127.

[56] GOUDARZI, H., AND PEDRAM, M. Hierarchical sla-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter. *IEEE Transactions on Cloud Computing 4*, 2 (June 2016), 222 – 236.

[57] GUÉROUT, T., GAOUA, Y., ARTIGUES, C., DA COSTA, G., LOPEZ, P., AND MONTEIL, T. Mixed integer linear programming for quality of service optimization in clouds. *Future Generation Computer Systems 71* (2017), 1–17.

[58] GUPTA, M. K., AND AMGOTH, T. Resource-aware virtual machine placement algorithm for iaas cloud. *The Journal of Supercomputing 74*, 1 (2018), 122–140.

[59] HAMMER, H. L., YAZIDI, A., AND BEGNUM, K. An inhomogeneous hidden markov model for efficient virtual machine placement in cloud computing environments. *Journal of Forecasting 36*, 4 (2017), 407–420.

[60] HERBST, N. R., KOUNEV, S., AND REUSSNER, R. Elasticity in cloud computing: What it is, and what it is not. In *10th International Conference on Autonomic Computing* (2013), 10th International Conference on Autonomic Computing, pp. 23–27.

[61] HERMENIER, F., LORCA, X., MENAUD, J.-M., MULLER, G., AND LAWALL, J. L. Entropy: a Consolidation Manager for Clusters. In *VEE 2009 - 5th International Conference on Virtual Execution Environments* (Washington, DC, United States, Mar. 2009), ACM, pp. 41–50.

[62] HINDMAN, B., KONWINSKI, A., ZAHARIA, M., GHODSI, A., JOSEPH, A. D., KATZ, R., SHENKER, S., AND STOICA, I. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (USA, 2011), NSDI'11, Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, pp. 295–308.

[63] HSIEH, S.-Y., LIU, C.-S., BUYYA, R., AND ZOMAYA, A. Y. Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers. *Journal of Parallel and Distributed Computing 139* (2020), 99–109.

[64] HUMMAIDA, A., PATON, N. W., AND SAKELLARIOU, R. Hierarchical decentralized architecture to enable adaptive scalable virtual machine migration. *Submitted to Concurrency and Computation: Practice and Experience (CCPE)* (2021).

[65] HUMMAIDA, A. R., PATON, N. W., AND SAKELLARIOU, R. Adaptation in cloud resource configuration: a survey. *Journal of Cloud Computing 5*, 1 (2016), 1–16.

[66] HUMMAIDA, A. R., PATON, N. W., AND SAKELLARIOU, R. Shdf - a scalable hierarchical distributed framework for data centre management. In *2017 16th International Symposium on Parallel and Distributed Computing (ISPDC)* (July 2017), 16th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 102–111.

[67] HUMMAIDA, A. R., PATON, N. W., AND SAKELLARIOU, R. Scalable virtual machine migration using reinforcement learning. *to be published in Journal of Grid Computing* (2021).

[68] HWANG, J., ZENG, S., WU, F. Y., AND WOOD, T. A component-based performance comparison of four hypervisors. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)* (2013), pp. 269–276.

[69] IDC. Cloud IT infrastructure spending continued to grow in Q1 2020 while spending on non-cloud environments saw double-digit declines, according to idc, June 2020.

[70] IQBAL, W., DAILEY, M. N., CARRERA, D., AND JANECEK, P. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems 26* (June 2011), 871–879.

[71] ISLAM, S., LEE, K., FEKETE, A., AND LIU, A. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering* (New York, NY, USA, 2012),

ICPE '12, Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, Association for Computing Machinery, pp. 85–96.

[72] JAIN, S. M. J. S. M. *Linux Containers and Virtualization A Kernel Perspective*, vol. 1. Apress, Berkeley, CA, 2020.

[73] JAMSHIDI, P., AHMAD, A., AND PAHL, C. Autonomic resource provisioning for cloud-based software. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (New York, NY, USA, 2014), SEAMS 2014, ACM, pp. 95–104.

[74] JAMSHIDI, P., SHARIFLOO, A. M., PAHL, C., METZGER, A., AND ESTRADA, G. Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution. In *2015 International Conference on Cloud and Autonomic Computing* (2015), pp. 208–211.

[75] JANGITI, S., AND VS, S. S. Emc2: Energy-efficient and multi-resource- fairness virtual machine consolidation in cloud data centres. *Sustainable Computing: Informatics and Systems 27* (2020), 100414.

[76] JEBA LEELIPUSHPAM, P. G., AND SHARMILA, J. Live VM migration techniques in cloud environment — a survey. In *2013 IEEE Conference on Information Communication Technologies* (2013), pp. 408–413.

[77] JIANG, C., WANG, Y., OU, D., LI, Y., ZHANG, J., WAN, J., LUO, B., AND SHI, W. Energy efficiency comparison of hypervisors. *Sustainable Computing: Informatics and Systems 22* (2019), 311–321.

[78] JIN, Y., BOUZID, M., KOSTADINOV, D., AND AGHASARYAN, A. Resource management of cloud-enabled systems using model-free reinforcement learning. *Annals of Telecommunications 74*, 9 (2019), 625–636.

[79] JUNG, G., HILTUNEN, M. A., JOSHI, K. R., SCHLICHTING, R. D., AND PU, C. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *International Conference on Distributed Computing Systems* (Washington, DC, USA, 2010), International Conference on Distributed Computing Systems, IEEE, pp. 62–73.

[80] KARDANI-MOGHADDAM, S., BUYYA, R., AND RAMAMOHANARAO, K. Adrl: A hybrid anomaly-aware deep reinforcement learning-based resource

scaling in clouds. *IEEE Transactions on Parallel and Distributed Systems 32*, 3 (March 2021), 514–526.

[81] KHAN, M. A., PAPLINSKI, A., KHAN, A. M., MURSHED, M., AND BUYYA, R. *Dynamic Virtual Machine Consolidation Algorithms for Energy-Efficient Cloud Resource Management: A Review*. Springer International Publishing, Cham, 2018, pp. 135–165.

[82] KHAN, T., TIAN, W., AND BUYYA, R. Machine learning (ml)-centric resource management in cloud computing: A review and future directions, 2021.

[83] KIM, S., AND CHOI, Y.-R. Constraint-aware VM placement in heterogeneous computing clusters. *Cluster Computing 23*, 1 (2020), 71–85.

[84] KLIAZOVICH, D., BOUVRY, P., AND KHAN, S. U. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing 62*, 3 (2012), 1263–1283.

[85] KOEHLER, M. An adaptive framework for utility-based optimization of scientific applications in the cloud. *Journal of Cloud Computing: Advances, Systems and Applications 3* (2014), 4.

[86] KULSHRESTHA, S., AND PATEL, S. An efficient host overload detection algorithm for cloud data center based on exponential weighted moving average. *International Journal of Communication Systems 34*, 4 (2021), e4708. e4708 dac.4708.

[87] KUSIC, D., KEPHART, J. O., HANSON, J. E., KANDASAMY, N., AND JIANG, G. Power and performance management of virtualized computing environments via lookahead control. In *Autonomic Computing ICAC* (Washington, DC, USA, Jun 2008), IEEE, pp. 3–23.

[88] LABIDI, T., MTIBAA, A., GAALOUL, W., TATA, S., AND GARGOURI, F. Cloud SLA modeling and monitoring. In *2017 IEEE International Conference on Services Computing (SCC)* (2017), pp. 338–345.

[89] LAMA, P., AND ZHOU, X. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing* (New York, NY, USA, 2012), ICAC '12, ACM, pp. 63–72.

[90] LEBRE, A., PASTOR, J., SIMONET, A., AND SÜDHOLT, M. Putting the next 500 vm placement algorithms to the acid test: The infrastructure provider viewpoint. *IEEE Transactions on Parallel and Distributed Systems 30*, 1 (Jan 2019), 204–217.

[91] LEHRIG, S., EIKERLING, H., AND BECKER, S. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures* (New York, NY, USA, 2015), QoSA '15, Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, Association for Computing Machinery, pp. 83–92.

[92] LIM, S.-H., SHARMA, B., NAM, G., KIM, E. K., AND DAS, C. R. Mdcsim: A multi-tier data center simulation, platform. In *2009 IEEE International Conference on Cluster Computing and Workshops* (2009), pp. 1–9.

[93] LIN, M., XI, J., BAI, W., AND WU, J. Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. *IEEE Access 7* (2019), 83088–83100.

[94] LIU, N., LI, Z., XU, J., XU, Z., LIN, S., QIU, Q., TANG, J., AND WANG, Y. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017), pp. 372–382.

[95] LORETI, D., AND CIAMPOLINI, A. A decentralized approach for virtual infrastructure management in cloud. *International Journal on Advances in Intelligent Systems 7*, 3/4 (2014), 507–518.

[96] LYNN, T., ROSATI, P., AND FOX, G. *Measuring the Business Value of Cloud Computing: Emerging Paradigms and Future Directions for Research*. Springer International Publishing, Cham, 2020, pp. 107–122.

[97] LYNN, T., ROSATI, P., LEJEUNE, A., AND EMEAKAROHA, V. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (2017), pp. 162–169.

[98] MANN, Z. A. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *ACM Computing Surveys 48*, 1 (Aug. 2015), 11:1–11:34.

[99] MANN, Z. Á., AND SZABÓ, M. Which is the best algorithm for virtual machine placement optimization? *Concurrency and Computation: Practice and Experience 29*, 10 (2017), e4083–n/a. e4083 cpe.4083.

[100] MASDARI, M., AND ZANGAKANI, M. Green cloud computing using proactive virtual machine placement: Challenges and issues. *Journal of Grid Computing 18*, 4 (2020), 727–759.

[101] MESBAHI, M. R., RAHMANI, A. M., AND HOSSEINZADEH, M. Reliability and high availability in cloud computing environments: a reference roadmap. *Human-centric Computing and Information Sciences 8*, 1 (2018), 20.

[102] MINAROLLI, D., MAZREKAJ, A., AND FREISLEBEN, B. Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing. *Journal of Cloud Computing 6*, 1 (2017), 4.

[103] MISHRA, S. K., MISHRA, S., BHARTI, S. K., SAHOO, B., PUTHAL, D., AND KUMAR, M. Vm selection using dvfs technique to minimize energy consumption in cloud system. In *2018 International Conference on Information Technology (ICIT)* (2018), pp. 284–289.

[104] MISHRA, S. K., PUTHAL, D., SAHOO, B., JAYARAMAN, P. P., JUN, S., ZOMAYA, A. Y., AND RANJAN, R. Energy-efficient VM-placement in cloud data center. *Sustainable Computing: Informatics and Systems 20* (2018), 48–55.

[105] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S., AND HASSABIS, D. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.

[106] MOENS, H., FAMAEY, J., LATRE, S., DHOEDT, B., AND TURCK, F. D. Design and evaluation of a hierarchical application placement algorithm in large scale clouds. In *IFIP/IEEE International Symposium on Integrated Network*

*Management* (2011), IFIP/IEEE International Symposium on Integrated Network Management, pp. 137–144.

[107] MOLA, O., AND BAUER, M. Towards cloud management by autonomic manager collaboration. *International Journal of Communications, Network and System Sciences 4*, 12A (2011), 790–802.

[108] MORENO-VOZMEDIANO, R., MONTERO, R. S., HUEDO, E., AND LLORENTE, I. M. Efficient resource provisioning for elastic cloud services based on machine learning techniques. *Journal of Cloud Computing 8*, 1 (2019), 5.

[109] MULLER-MERBACH, H. Heuristics and their design: a survey. *European Journal of Operational Research 8*, 1 (September 1981), 1–23.

[110] NADGOWDA, S., SUNEJA, S., BILA, N., AND ISCI, C. Voyager: Complete container state migration. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (2017), pp. 2137–2142.

[111] NATHUJI, R., KANSAL, A., AND GHAFFARKHAH, A. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European Conference on Computer Systems* (New York, NY, USA, 2010), EuroSys '10, ACM, pp. 237–250.

[112] NOURI, S. M. R., LI, H., VENUGOPAL, S., GUO, W., HE, M., AND TIAN, W. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Generation Computer Systems 94* (2019), 765 – 780.

[113] PADALA, P., SHIN, K. G., ZHU, X., UYSAL, M., WANG, Z., SINGHAL, S., MERCHANT, A., AND SALEM, K. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (New York, NY, USA, 2007), EuroSys '07, Association for Computing Machinery, pp. 289–302.

[114] PANTAZOGLOU, M., TZORTZAKIS, G., AND DELIS, A. Decentralized and energy-efficient workload management in enterprise clouds. *IEEE Transactions on Cloud Computing 4*, 2 (April 2016), 196–209.

[115] PENG, Z., LIN, J., CUI, D., LI, Q., AND HE, J. A multi-objective trade-off framework for cloud resource scheduling based on the deep q-network algorithm. *Cluster Computing 23*, 4 (2020), 2753–2767.

[116] PIETRI, I., AND SAKELLARIOU, R. Mapping virtual machines onto physical machines in cloud computing: A survey. *ACM Computing Surveys 49*, 3 (Oct. 2016).

[117] QUESNEL, F., LÈBRE, A., AND SÜDHOLT, M. Cooperative and reactive scheduling in large-scale virtualized platforms with dvms. *Concurrency and Computation: Practice and Experience 25*, 12 (2013), 1643–1655.

[118] RAHMANIAN, A. A., HORRI, A., AND DASTGHAIBYFARD, G. Toward a hierarchical and architecture-based virtual machine allocation in cloud data centers. *International Journal of Communication Systems 31*, 4 (2018), e3490. e3490 IJCS-17-0412.R1.

[119] RAMAMOORTHY, S., RAVIKUMAR, G., SARAVANA BALAJI, B., BALAKRISHNAN, S., AND VENKATACHALAM, K. Mcamo: multi constraint aware multi-objective resource scheduling optimization technique for cloud infrastructure services. *Journal of Ambient Intelligence and Humanized Computing 12*, 6 (2021), 5909–5916.

[120] RAO, J., BU, X., XU, C.-Z., AND WANG, K. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems* (2011), pp. 45–54.

[121] RAO, J., BU, X., XU, C.-Z., WANG, L., AND YIN, G. Vconf: A reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th International Conference on Autonomic Computing* (New York, NY, USA, 2009), ICAC '09, Association for Computing Machinery, pp. 137–146.

[122] REN, H., WANG, Y., XU, C., AND CHEN, X. Smig-rl: An evolutionary migration framework for cloud services based on deep reinforcement learning. *ACM Transactions on Internet Technology 20*, 4 (Oct. 2020).

[123] SAADI, Y., AND EL KAFHALI, S. Energy-efficient strategy for virtual machine consolidation in cloud environment. *Soft Computing 24*, 19 (2020), 14845–14859.

[124] SAIF, M. A. N., NIRANJAN, S. K., AND AL-ARIKI, H. D. E. Efficient autonomic and elastic resource management techniques in cloud environment: taxonomy and analysis. *Wireless Networks 27*, 4 (2021), 2829–2866.

[125] SATHYA SOFIA, A., AND GANESHKUMAR, P. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using nsga-ii. *Journal of Network and Systems Management 26*, 2 (2018), 463–485.

[126] SEDAGHAT, M., HERNÁNDEZ-RODRIGUEZ, F., ELMROTH, E., AND GIRDZIJAUSKAS, S. Divide the task, multiply the outcome: Cooperative vm consolidation. In *IEEE International Conference on Cloud Computing Technology and Science* (Washington, DC, USA, Aug 2014), IEEE International Conference on Cloud Computing Technology and Science, IEEE, pp. 300–305.

[127] SHAHIDINEJAD, A., GHOBAEI-ARANI, M., AND MASDARI, M. Resource provisioning using workload clustering in cloud computing environment: a hybrid approach. *Cluster Computing 24*, 1 (2021), 319–342.

[128] SHAW, R., HOWLEY, E., AND BARRETT, E. Applying reinforcement learning towards automating energy efficient virtual machine consolidation in cloud data centers. *Information Systems* (2021), 101722.

[129] SHEN, Z., SUBBIAH, S., GU, X., AND WILKES, J. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing* (New York, NY, USA, 2011), SOCC '11, ACM, pp. 5:1–5:14.

[130] SHI, W., AND HONG, B. Towards profitable virtual machine placement in the data center. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing* (2011), pp. 138–145.

[131] SILVA FILHO, M. C., MONTEIRO, C. C., INÁCIO, P. R., AND FREIRE, M. M. Approaches for optimizing virtual machine placement and migration in cloud environments: A survey. *Journal of Parallel and Distributed Computing 111* (2018), 222–250.

[132] SNIEZYNSKI, B., NAWROCKI, P., WILK, M., JARZAB, M., AND ZIELINSKI, K. Vm reservation plan adaptation using machine learning in cloud computing. *Journal of Grid Computing 17*, 4 (2019), 797–812.

[133] STILLWELL, M., SCHANZENBACH, D., VIVIEN, F., AND CASANOVA, H. Resource allocation using virtual clusters. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (2009), pp. 260–267.

[134] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*, vol. 1. Cambridge:MIT press, 1998.

[135] SVÄRD, P., HUDZIA, B., WALSH, S., TORDSSON, J., AND ELMROTH, E. Principles and performance characteristics of algorithms for live vm migration. *SIGOPS Oper. Syst. Rev. 49*, 1 (Jan. 2015), 142–155.

[136] TCHANA, A., PALMA, N. D., SAFIEDDINE, I., HAGIMONT, D., DIOT, B., AND VUILLERME, N. Software consolidation as an efficient energy and cost saving solution for a saas/paas cloud model. In *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings* (Berlin, Heidelberg, 2015), L. J. Träff, S. Hunold, and F. Versaci, Eds., Springer Berlin Heidelberg, pp. 305–316.

[137] THIAM, C., AND THIAM, F. Energy efficient cloud data center using dynamic virtual machine consolidation algorithm. In *Business Information Systems* (Cham, 2019), W. Abramowicz and R. Corchuelo, Eds., Springer International Publishing, pp. 514–525.

[138] TIGHE, M., KELLER, G., BAUER, M., AND LUTFIYYA, H. Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management. Network and service management (CNSM), 2012 8th international conference and 2012 workshop on systems virtualization management (SVM), pp. 385–392.

[139] TULI, S., GILL, S. S., XU, M., GARRAGHAN, P., BAHSOON, R., DUSTDAR, S., SAKELLARIOU, R., RANA, O., BUYYA, R., CASALE, G., AND JENNINGS, N. R. HUNTER: AI based holistic resource management for sustainable cloud computing. *CoRR abs/2110.05529* (2021).

[140] ULLAH, A., LI, J., SHEN, Y., AND HUSSAIN, A. A control theoretical view of cloud elasticity: taxonomy, survey and challenges. *Cluster Computing 21*, 4 (2018), 1735–1764.

[141] VAEZI, M., AND ZHANG, Y. *Virtualization and Cloud Computing*. Springer International Publishing, Cham, 2017, pp. 11–31.

[142] VAN, H. N., TRAN, F. D., AND MENAUD, J.-M. SLA-aware virtual resource management for cloud infrastructures. In *IEEE International Conference on Computer and Information Technology* (Washington, DC, USA, 2009), vol. 02, IEEE International Conference on Computer and Information Technology, IEEE, pp. 357–362.

[143] VAN EYK, E., IOSUP, A., SEIF, S., AND THÖMMES, M. The spec cloud group's research vision on faas and serverless architectures. In *Proceedings of the 2nd International Workshop on Serverless Computing* (New York, NY, USA, 2017), WoSC '17, Association for Computing Machinery, pp. 1–4.

[144] VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, Association for Computing Machinery.

[145] WANG, B., SONG, Y., CUI, X., AND CAO, J. Performance comparison between hypervisor- and container-based virtualizations for cloud users. In *2017 4th International Conference on Systems and Informatics (ICSAI)* (2017), pp. 684–689.

[146] WANG, Z., CHEN, Y., GMACH, D., SINGHAL, S., WATSON, B. J., RIVERA, W., ZHU, X., AND HYSER, C. D. Appraise: application-level performance management in virtualized server environments. *IEEE Transactions on Network and Service Management 6*, 4 (2009), 240–254.

[147] WATKINS, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, 1989.

[148] WEERASIRI, D., BARUKH, M. C., BENATALLAH, B., SHENG, Q. Z., AND RANJAN, R. A taxonomy and survey of cloud resource orchestration techniques. *ACM Comput. Surv. 50*, 2 (may 2017).

[149] WITANTO, J. N., LIM, H., AND ATIQUZZAMAN, M. Adaptive selection of dynamic vm consolidation algorithm using neural network for cloud resource management. *Future Generation Computer Systems 87* (2018), 35–42.

[150] WU, Y., LEI, L., WANG, Y., SUN, K., AND MENG, J. Evaluation on the security of commercial cloud container services. In *Information Security* (Cham, 2020), W. Susilo, R. H. Deng, F. Guo, Y. Li, and R. Intan, Eds., Springer International Publishing, pp. 160–177.

[151] WUHIB, F., STADLER, R., AND SPREITZER, M. Dynamic resource allocation with management objectives: implementation for an openstack cloud. *IEEE Transactions on Network and Service Management 9*, 2 (2012), 213–225.

[152] WUHIB, F., STADLER, R., AND SPREITZER, M. A gossip protocol for dynamic resource management in large cloud environments. *IEEE Transactions on Network and Service Management 9*, 2 (2012), 213–225.

[153] XU, H., LIU, Y., WEI, W., AND XUE, Y. Migration cost and energy-aware virtual machine consolidation under cloud environments considering remaining runtime. *International Journal of Parallel Programming 47*, 3 (2019), 481–501.

[154] YADAV, M. P., ROHIT, AND YADAV, D. K. Resource provisioning through machine learning in cloud services. *Arabian Journal for Science and Engineering* (2021).

[155] YADAV, R., ZHANG, W., LI, K., LIU, C., SHAFIQ, M., AND KARN, N. K. An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center. *Wireless Networks 26*, 3 (2020), 1905–1919.

[156] YAN, Y., ZHANG, B., AND GUO, J. An adaptive decision making approach based on reinforcement learning for self-managed cloud applications. In *2016 IEEE International Conference on Web Services (ICWS)* (2016), pp. 720–723.

[157] YING, C., LI, B., KE, X., AND GUO, L. Raven: Scheduling virtual machine migration during datacenter upgrades with reinforcement learning. *Mobile Networks and Applications* (2020).

[158] ZAHEDI FARD, S. Y., AHMADI, M. R., AND ADABI, S. A dynamic vm consolidation technique for qos and energy consumption in cloud environment. *The Journal of Supercomputing 73*, 10 (2017), 4347–4368.

[159] ZHANG, Q., CHEN, H., SHEN, Y., MA, S., AND LU, H. Optimization of virtual resource management for cloud applications to cope with traffic burst. *Future Generation Computer Systems 58* (2016), 42 – 55.

[160] ZHANG, Q., CHENG, L., AND BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications 1*, 1 (2010), 7–18.

[161] ZHANI, M., CHERITON, D. R., ZHANG, Q., SIMON, G., AND BOUTABA, R. Vdc planner: Dynamic migration-aware virtual data center embedding for clouds. In *IEEE International Symposium on Integrated Network Management* (Washington, DC, USA, 2013), IEEE, pp. 18–25.

[162] ZHAO, W., PENG, Y., XIE, F., AND DAI, Z. Modeling and simulation of cloud computing: A review. In *2012 IEEE Asia Pacific Cloud Computing Congress (APCloudCC)* (2012), pp. 20–24.

[163] ZHENG, S., ZHU, G., ZHANG, J., AND FENG, W. Towards an adaptive human-centric computing resource management framework based on resource prediction and multi-objective genetic algorithm. *Multimedia Tools and Applications* (2015), 1–18.

[164] ZHENG, X., AND XIA, Y. Exploring mixed integer programming reformulations for virtual machine placement with disk anti-colocation constraints. *Performance Evaluation 135* (2019), 102035.

[165] ZHU, X., YOUNG, D., WATSON, B. J., WANG, Z., ROLIA, J., SINGHAL, S., MCKEE, B., HYSER, C., GMACH, D., GARDNER, R., CHRISTIAN, T., AND CHERKASOVA, L. 1000 islands: Integrated capacity and workload management for the next generation data center. In *International Conference on Autonomic Computing* (Washington, DC, USA, Jun 2008), International Conference on Autonomic Computing, IEEE, pp. 172–181.

[166] ZOLFAGHARI, R., SAHAFI, A., RAHMANI, A. M., AND REZAEI, R. Application of virtual machine consolidation in cloud computing systems. *Sustainable Computing: Informatics and Systems 30* (2021), 100524.

[167] ZUO, L., SHU, L., DONG, S., ZHU, C., AND ZHOU, Z. Dynamically weighted load evaluation method based on self-adaptive threshold in cloud computing. *Mobile Networks and Applications* (2016), 1–15.