Journal of Network and Computer Applications 167 (2020) 102740

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

An automated implementation of hybrid cloud for performance evaluation of distributed databases

Yaser Mansouri [*], Victor Prokhorenko, M. Ali Babar

Centre for Research on Engineering Software Technologies (CREST) Lab, School of Computer Science, The University of Adelaide, Adelaide, Australia

ARTICLE INFO

Keywords:
Hybrid cloud
Cloud bursting
Distributed databases
WireGuard VPN
Throughput
Read and write latency

ABSTRACT

A Hybrid cloud is an integration of resources between private and public clouds. It enables users to horizontally scale their on-premises infrastructure up to public clouds in order to improve performance and cut up-front investment cost. This model of applications deployment is called cloud bursting that allows data-intensive applications especially distributed database systems to have the benefit of both private and public clouds. In this work, we present an automated implementation of a hybrid cloud using (i) a robust and zero-cost Linux-based VPN to make a secure connection between private and public clouds, and (ii) Terraform as a software tool to deploy infrastructure resources based on the requirements of hybrid cloud. We also explore performance evaluation of cloud bursting for six modern and distributed database systems on the hybrid cloud spanning over local OpenStack and Microsoft Azure. Our results reveal that MongoDB and MySQL Cluster work efficient in terms of throughput and operations latency if they burst into a public cloud to supply their resources. In contrast, the performance of Cassandra, Riak, Redis, and Couchdb reduces if they significantly leverage their required resources via cloud bursting.

1. Introduction

Cloud computing is the mainstream delivery of on-demand and easy-to-use services such as computing, networking, storage, databases, and software over the Internet (Buyya et al., 2009). It provides a pay-per-usage model in which consumers typically pay for cloud services they use. Cloud computing comes traditionally into two models (Armbrust et al., 2010): *public* and *private*. A public cloud provides computing, storage and networking resources to the general public over Internet while a private cloud facilitates resources of on-premises infrastructure for the dedicated use of a specific organization.

A hybrid cloud is a seamless integration of public and private clouds to take the best of both worlds (Rimal et al., 2009). It enables *cloud bursting* in which applications initially leverage private cloud and burst into a public cloud when private resources are not enough to provision under spiking workload. A hybrid cloud significantly benefits its owner in terms of security in compliance with the location of sensitive data, availability, reliability and cost reduction. However, *construction of hybrid clouds* and *deployment of applications on top of them* are not trivial tasks. We investigate both tasks in this work.

In respect to the hybrid cloud implementation, the first issue we intend to address is a secure, robust and cost-effective connection between public and private clouds. Construction of hybrid clouds demands a secure connection between computational resources scattered across private and public clouds under different administrative domains. It means that different virtual machines/nodes in sub-networks need to be connected. This connectivity implies that two separate sets of IP ranges should be connected to make an automated resource provisioning across private and public clouds. One viable solution is to allocate public IPs to VMs to make connections between clouds over Internet. Provided public IPs in a specific range for a private cloud is infeasible in most cases. Moreover, hybrid clouds require a secure connection channel especially for mission-critical systems in which secure data is transmitted over Internet.

Network virtualization techniques allow a secure connection channel between private and public clouds through building an overlay network over the Internet. Virtual Private Networks (VPNs) provide such overlay network to make a secure connection between VMs in the hybrid cloud. The well-known cloud providers offer different VPN connectivity options to make a secure connection across clouds (Han-Haihong et al., 2011). Such VPN options may not be compatible, robust (observed for Azure VPN), and suffer from high monetary cost. In con-

[*] Corresponding author.
E-mail address: aser.mansouri@adelaide.edu.au (Y. Mansouri).

https://doi.org/10.1016/j.jnca.2020.102740
Received 14 January 2020; Received in revised form 3 May 2020; Accepted 3 June 2020
Available online 12 June 2020
1084-8045/© 2020 Elsevier Ltd. All rights reserved.

trast, WireGuard[1] was initially introduced in 2016 as a Linux kernel-based VPN that provides a *flexible*, *robust*, *interoperable*, *secure* and *zero-cost connection* between private and public clouds according to individual needs.

In regard to the hybrid cloud implementation, the second issue we want to figure out is to deploy infrastructure and applications in an automated manner. Automation of a hybrid cloud implementation is the key concept to deploy infrastructure resources, to make a secure connection between clouds, and to route data among sub-networks across clouds. Such automation allows users to quickly exploit resources with different flavors, reduces human's work, and consistently creates an environment to deploy the desired big data applications. We leveraged Terraform[2] as a software tool to provide an automated implementation of the hybrid cloud based on the required VMs size, VMs number, sub-network features (i.e., ingress and egress port number) and so forth.

Today's IT systems may benefit from hybrid clouds to boost performance, manage business velocity, avoid data lock-in, and keep sensitive data on-premise and insensitive data in public clouds. The performance of such systems depends on the cloud-based distributed databases that might be different in serving read and write operations, and managing data replication, data sharding, and data consistency. These databases are classified into relational and NoSQL (Mansouri et al., 2017). Relational databases have pre-defined schemas and possess a mature market. By contrast, NoSQL databases have a schema-less data model and compromise strong consistency for better performance.

Due to ever-increasing demands in data Volume, Variety, and Velocity (i.e., 3 Vs in big data), relational databases might not be applicable to such data and NoSQL databases seem a viable solution (HanHaihong et al., 2011). Currently, there are more than 225 NoSQL databases[3] widely used in large-scale companies like Facebook, Twitter, Amazon, and Google. These companies provide infrastructure for small- and medium-size businesses to run NoSQL databases. Nevertheless, such businesses do not completely store their data on public infrastructure (provided by these companies) and exploit their private infrastructure to achieve their needs in performance and security aspects. Due to increasing deployment of NoSQL databases in different industry domains like e-commerce, IoT, blockchain, social networks, to name a few, the evaluation of cloud bursting at the level of NoSQL databases not only attracts research attention but is also useful from industry perspective to investigate the impact of cloud bursting on distributed databases.

Integrating private and public clouds to build a hybrid cloud was initially conducted in the academic domain to evaluate the performance of different applications (Toosi et al., 2018; Calheiros et al., 2011). Due to potential benefits of hybrid clouds, the well-known cloud providers offer developers rich platforms to run applications in an on-premises environment and deliver the same services as in the public clouds. In fact, such platforms make easy to build up a hybrid cloud. Microsoft Azure released Azure Stack Development Kit (ASDK)[4] in 2017, following *Outposts*[5] and *Anthos*[6] by AWS and Google in 2018. Neither commercial nor academic projects reveal how they support hybrid clouds. None of these projects have revealed how to build up a general architecture for hybrid clouds. Also, due to the nature of commercial products, cloud providers only offer these products as black-boxes and developers can deploy them as a hybrid cloud in conjunction with public clouds.

There is a large body of literature on deployment of data-intensive applications on hybrid clouds (Toosi et al., 2018; Calheiros et al., 2012; Vecchiola et al., 2012; Zhou et al., 2019). These studies focused on reduction of execution time for data-intensive applications within constrained time, monetary cost, or both. Several of these studies measured performance evaluation in hybrid clouds. However, they neither leveraged a cost-free, secure, and resilient VPN to build a hybrid cloud, nor evaluated distributed databases on the hybrid clouds. By contrast, several studies evaluated the performance of distributed databases either on the public clouds (Kuhlenkamp et al., 2014; Klein et al., 2015) or private clouds (Abramova and Bernardino, 2013; Lima et al., 2016; Li and Manoharan, 2013). None of these studies have specifically assessed the performance impact of cloud bursting on distributed databases running in hybrid clouds to comprehend how well these databases work on this model of clouds. In contrast to the public and private clouds offering all resources at the same datacenter, we strongly believe that distance between the private and public cloud datacenters has significant effects on the performance of distributed databases. Therefore, all these gaps discussed above motivate us to make the fallowing contributions.

- We present different usage models of hybrid clouds, and for each model we identify purpose, key challenges, optimization domain, features and applicability.
- We automate the implementation of the hybrid cloud using Terraform as an automation tool to provision and manage the infrastructure of clouds. In our implementation, we use WireGuard, as Linux kernel-based VPN, to make a secure, robust, and zero-cost connection between public and private clouds.
- We conduct experimental cloud-bursting evaluation at the database level and report on the details of our experience with this implementation from an industry view point.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 delineates different usage models of hybrid clouds, following by an automated implementation of the hybrid cloud in Section 4. Section 5 describes general features of distributed databases and workload setup. Performance evaluation is discussed in Section 6. Finally, Section 7 concludes the paper and outlines the future research directions.

2. Related work

Hybrid cloud, benefit and challenges: A Hybrid cloud integrates on-premises infrastructure (i.e., a private cloud) and a public cloud like Microsoft Azure and Amazon Web Services (AWS). It gives an organization an ability to scale its on-premises infrastructure up to handle workload spikes that demand more resources than available locally. A hybrid cloud brings the "best of both worlds" in the following features.

Security: A Hybrid cloud enables organizations to split the data into *sensitive* and *insensitive* for storing respectively in the private and public clouds. *Resilience*: As the ability to guarantee an acceptable level of service in the case of faults and challenges to normal operations. Resilience of a hybrid cloud depends on the infrastructure of clouds (guaranteed by cloud providers) and the network connection between clouds (guaranteed by VPN connection). To make a resilient and robust connection between clouds, we exploit WireGuard. *Scalability and Reliability*: A hybrid cloud allows organizations to add/remove resources to/from their pool of resources as workload changes. It also offers higher reliability in the case of natural disasters as users can recover data from a backup if data was replicated in both clouds. *Cost saving*: A hybrid cloud mitigates capital expenditures to handle short-term spikes in workloads that require resources beyond the available ones in a private cloud. Nevertheless, a connection between private and public clouds can be costly. These features can be achieved through a secure, resilient and cost-free VPN (e.g.,WireGuard) that connects the private and public clouds involving in the hybrid cloud.

Being no exception, a hybrid cloud arises several challenges in two dimensions. The first one is implementation of hybrid clouds for which robust, automated, secure and cost-effective connection, network routing, and provisioning cloud infrastructure should be figured out. The

[1] WireGuard: https://www.wireguard.com/.

[2] Terraform: https://www.terraform.io/.

[3] NoSQL databases: http://nosql-database.org/.

[4] ASDK:https://azure.microsoft.com/en-us/overview/azure-stack/.

[5] Outposts: https://aws.amazon.com/outposts/.

[6] Anthos: https://cloud.google.com/anthos/.

Table 1
Comparison of relevant studies with our work.

Paper	Application	Cloud	Network connection	Cloud bursting	Evaluation	Objectives
Toosi et al. (2018)	BoT	Hybird	Public VPN	Yes	Imp.	Providing deadline requirements for data-intensive applications
Calheiros et al. (2012)	BoT	Hybird	NS	Yes	Imp.	Increasing the capacity of Desktop Grids via Cloud
Vecchiola et al. (2012)	BoT	Hybird	NS	Yes	Imp.	Reduction in execution time via Ankeka and Cloud
Tuli et al. (2020)	BoT	Hybrid	Public VPN	Yes	Imp.	Reduction in bandwidth consumption via considering sharing files
Li et al. (2018)	OSN	Hybrid	IPsec tunnel	Yes	Imp.	Reduction in waiting and response times in a private cloud
Moschakis and Karatza (2015)	BoT	Hybrid	NA	Yes	Sim.	Optimization of performance and cost via scheduling tasks
Abdi et al. (2017)	BoT	Hybrid	NA	Yes	Sim.	Minimizing the total cost of applications in cloud federation
Zhou et al. (2019)	Workflow	Hybrid	NA	Yes	Sim.	Optimization monetary cost and makespan of workflow applications
Rabl et al. (2012)	DDB	Private	NA	No	Imp.	Performance evaluation of Cassandra, HBase, Redis, Voldemort, VoltDB, MySQL
Kuhlenkamp et al. (2014)	NSDB	Public	NA	No	Imp.	Scalability and elasticity evaluation of Cassandra and HBase
Li and Manoharan (2013)	NSDB	Private	NA	No	Imp.	Evaluation of MongoDB, RavenDB, CouchDB, Cassandra, Hypertable, Couchbase, and MySQL in performance
Klein et al. (2015)	NSDB	Public	NA	No	Imp.	Evaluation of MongoDB, Cassandra, and Riak in consistency support
Abramova and Bernardino (2013)	NSDB	Private	NA	No	Imp.	A comparison between Cassandra and MongoDB in performance
This Work	DDB	Hybrid	WireGuard	Yes	Imp.	Implementation of a hybrid cloud in the network level and evaluation of DDBs in the context of cloud bursting

Abbreviations: BoT: Bag of Task; OSN: Online Social Network; DDB: Distributed Databases; NSDB: No-SQL Databases; NS: Not Specified; NA: Not Applicable; Imp: Implementation; Sim: Simulation.

second dimension is the deployment of applications in hybrid clouds for which automated resource discovery and optimized Quality of Service (QoS)– such as monetary cost and response time for running applications on a hybrid cloud– should be addressed. In summary, these challenges depend on the usage models of hybrid clouds discussed later. There is a plethora of literature that addresses these challenges in the context of the hybrid clouds. To position our work, we divide related work into the following subsections.

Hybrid cloud implementation: Due to strong desire to exploit hybrid clouds by IT businesses, recently the well-known cloud providers enable deployment of their native services to on-premises infrastructure. This brings a potential capability to implement a truly consistent hybrid cloud through VPN. In this respect, Microsoft Azure offered Azure Stack Development Kit (ASDK) in 2017. Due to cut-throat competition among cloud providers, AWS and Google respectively offered *Outposts* and *Anthos* in 2018. In spite of such benefits, these products are not lightweight in hardware requirements and incur heavy costs through using VPNs to build hybrid cloud.

Before releasing these commercial products, researchers have conducted rich research on hybrid clouds. They evaluated their proposed algorithms, policy, and methods through simulation and implementation. Toosi et al. (2018) recently set up a hybrid cloud including Microsoft Azure and two PC workers to evaluate their data-aware resource provisioning algorithms. Tuli et al. (2020) designed dynamic resource provisioning and task scheduling algorithms and evaluated on the same setup of a hybrid cloud in Toosi et al. (2018) but with different VM flavors. Calheiros et al. (2012) and Vecchiola et al. (2012) have conducted almost similar setup to evaluate an algorithm leveraging dynamic resources to meet the deadline requirements of Bag-of-Task. Li et al. (2018) proposed a cost-aware job scheduling approach based on queuing theory in hybrid clouds in a static setup using a public VPN. Differently, Loreti and Ciampolini (2015) implemented a software layer on top of hybrid cloud infrastructure to dynamically deploy and scale virtual clusters.

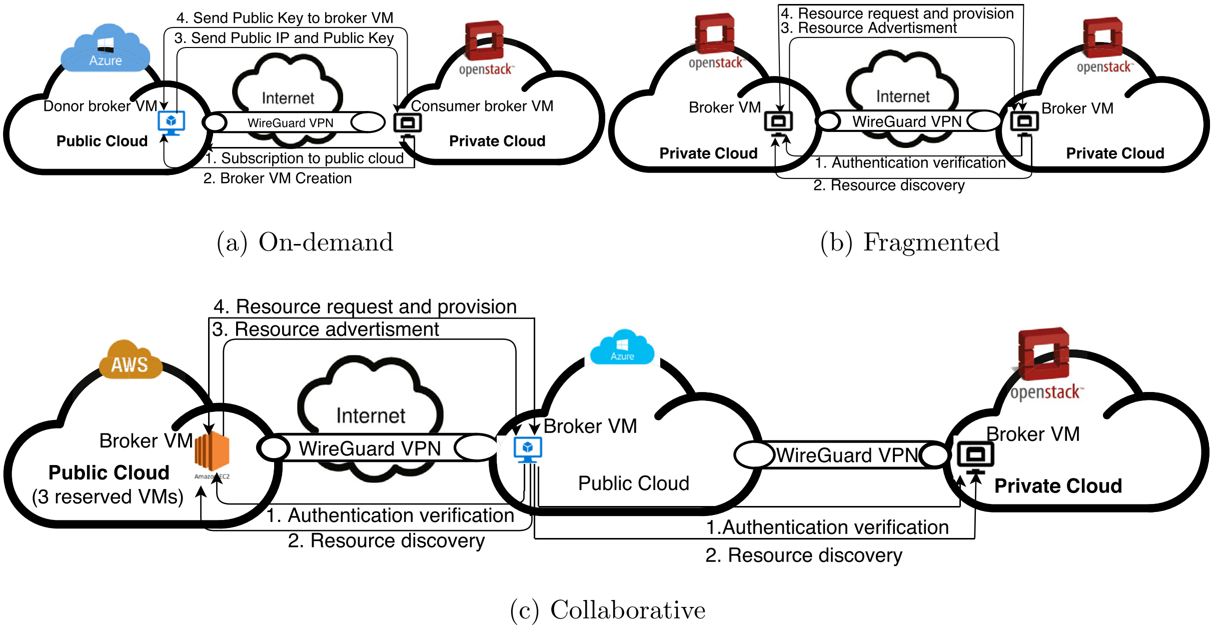
The performance impact of cloud bursting on distributed database systems: Cloud bursting allows an application to run in a private cloud and burst into a public cloud in case the demand for computing and storage resources spikes. One suitable candidate for such deployment is distributed databases which are different in data model, data sharding, data replication and consistency.

NoSQL databases tend to be a better solution for modern big data applications. They represented themselves as alternatives that can store big data characterized by 3 Vs: *volume*, *veracity* and *variety*. Due to a variety of NoSQL databases with different features, it is crucial to evaluate the performance impact of cloud bursting on such databases. Initially in Li and Manoharan (2013), the performance of some distributed databases have been investigated for read, write, delete, and instantiate operations. Rabl et al. (2012) evaluated four NoSQL databases (Voldermort,[7] HBase,[8] Cassandra[9] , and Redis,[10] and two relational databases (MySQL cluster[11] and VoltDB[12]) with focus on performance and scalability. Kuhlenkamp et al. (2014) conducted experiments on NoSQL databases (HBase and Cassandra) in terms of scalability and elasticity. In these studies, all experimental evaluations have been conducted on either a public or private cloud. Thus, they neither deal with the implementation of a hybrid cloud nor evaluate distributed databases in this model of cloud.

Besides distributed databases, the other data intensive applications have been recently evaluated on hybrid clouds. Xu et al. (2017) developed a cost and energy aware data placement method for privacy-aware big data applications. Moschakis and Karatza (2015), Abdi et al. (2017) and Xiong et al. (2016) investigated time- and cost-constrained algorithms to handle cloud-bursting using simulators like Cloud-Sim. Zhou et al. (2019) proposed an approach that optimizes the monetary cost of scheduling workflows under constrained time, and then extended it to optimize both makespan and monetary cost for workflow scheduling. They achieved up to 100% cost saving and an effective cost-makespan trade-off in comparison to the competing approaches through a simulation platform. In contrast to our work, these studies all have investigated the effects of methods in a simulated testbed.

In comparison to the discussed studies in Table 1, our work is different in two aspects. (i) we have automated the implementation of hybrid cloud at the network level using Terraform as an automation tool and WireGuard as a secure, robust, and zero-cost VPN. This automated implementation of the hybrid cloud allows us to dynamically, consistently and repeatedly (with the least human interference) exploit infrastructure resources based on the desired features such as VMs number, VMs size, VMs region, Network features, database installation, database cluster configuration, and so on. (ii) We have evaluated the impact of cloud bursting on the performance of distributed databases.

[7] Voldermort: https://www.project-voldemort.com/voldemort/.
[8] HBase:https://hbase.apache.org/.
[9] Cassandra: http://cassandra.apache.org/.
[10] Redis:https://redis.io/.
[11] MySQL:https://www.mysql.com/.
[12] VoltDB:https://www.voltdb.com/.



(a) On-demand

(b) Fragmented

(c) Collaborative

Fig. 1. Different usage models of hybrid clouds. (a) On-demand, (b) Fragmented, and (c) Collaborative.

3. Hybrid cloud usage models

The usage models of a hybrid cloud define resource sharing pattern of private and public clouds. The connection is influenced by the authentication process and the amount of shared resources in a hybrid cloud. In our models, a *consumer* cloud requests resources and a *donor* cloud shares its resources with the consumer cloud. Obviously, consumer and donor clouds can be either a private cloud or a public cloud. In the following, we propose different usage models of hybrid clouds.

3.1. On-demand usage model

This model is known as *cloud bursting* and makes a secure connection between a private cloud and a public cloud, where private and public clouds we respectively call *consumer* and *donor*. When a data-intensive application running on a private cloud and requiring a resource (e.g., a VM instance) the application provider subscribes to a public cloud and creates it in the public cloud using either automation tools such as Terraform,[13] Ansible,[14] or APIs provided by the public cloud. After that, the private cloud makes a connection with the public cloud through WireGuard. Fig. 1a shows a summary of collaboration between two clouds in three steps: subscription to a public cloud, creation of resource, and connection between private and public clouds by exchanging public keys and VM's public IP in the public cloud. This model can influence the shared resources in terms of security aspect since data should be transferred to a public cloud. In contrast, it does not impede the amount of resources to share because public clouds offer an illusion of infinite resources.

3.2. Fragmented usage model

This model makes a connection between two or more private clouds and is suitable for mission-critical systems that possess a private cloud. The amount of resources depends on a sub-system size, where in most cases the smaller sub-system demands resources from the bigger one. As shown in Fig. 1b, the process of deploying resources across two private clouds consists of authentication verification between two clouds, dynamic resource discovery, advertisement of resources by donor cloud, resource request and provision.

In this model of usage, private clouds deploying by different sub-systems can be pre-configured in network connection; as a result it faces less security concerns. The exploitation of such usage model in a mobile environment leads to as sub-systems/devices are approaching more, the amount of resources available is more. In contrast, as they are moving away the amount of resources available is less. Hence, a challenging problem of this environment is to share resources on-the-fly between different sub-systems/devices due to their mobility. This implies that, in contrast to the previous model, we need to consider connection and disconnection of network and thus dynamic discovery of resources is a must.

3.3. Collaborative usage model

This model makes a connection between different models of clouds (i.e., private and public) authorized under a single or multiple collaborating organization. As depicted in Fig. 1c, an organization provides resources from different cloud providers (e.g., Microsoft Azure, AWS, and Google) as well as private clouds. We may require making network connections between two public clouds, two private clouds, or one private and one public cloud. In the collaborative usage model, the resource sharing between two public clouds occurs when the reserved resources are free in the public cloud while the organization requires resources more than that reserved in the consumer cloud. It is worth noting that public providers offer reserved computation and database resources in terms of one and three years. These resources are cheaper than the on-demand resources but can be wasted if not used in the given time. This encourages different parts of an organization to share their reserved resources between co-workers. In this model, the connection between private and public clouds can be a specific case of the first model while the amount of resources providing by the public cloud is limited. In summary, the collaborative usage model inherits limitations of the first two models, i.e., security concerns and constrained resources, and in contrast to the second usage model it is a static model. Table 2 summarizes the discussed usage models of hybrid clouds in purpose, key challenges, optimization domains, features, and applicability.

[13] Terraform: https://www.terraform.io/.

[14] Ansible: https://www.ansible.com/.

Table 2
A comparison of different usage models of hybrid Clouds.

	On-demand	Fragmented	Collaborative
Purpose	To gain temporary access to external resources	Dynamically discover and use shared resources based on geographic location and network reachability	To use shared pool of resources within a single or multiple collaborating organization
Key challenges	Data privacy	Abrupt resource disconnection	Multitude of incompatible clouds
Secure communication	Automated resource discovery	Secure Communication	
Network routing	Network routing	Network routing	
	Secure communication		
Optimization domains	Performance	Latency	Performance
Latency	Bandwidth	Latency	
Bandwidth	–	Bandwidth	
Monetary cost	–	–	
Features	No upfront cost	Up-front cost	Depend on combination type of clouds
No data-lock in	Data lock-in	–	
A variety of resource types	Limited resource type	–	
Applicability	Business organizations	Critical-mission systems	Depend on combination of cloud types

4. An automated implementation of hybrid cloud

Among the usage models of hybrid clouds discussed above, we select on-demand usage model as one of the most conventional deployment of hybrid clouds.[15] Two challenging issues relating to the implementation of this model is discussed below.

Connectivity issues in hybrid clouds: A secure connectivity[16] is one of the integral parts of the deployment of a hybrid cloud since virtual machines (VMs) are scattered across networks with different administration domains. One possible solution is the allocation of public IP addresses to the deployed VMs in the private cloud. However, allocation of public IPs to VMs in a private cloud is waste of resources as if there is no limitation in allocation of public IPs by an IT business. Furthermore, VMs in a private cloud usually operate behind organizational firewalls and network address translation (NAT), which hinder developers to make connections via public IPs.

However, network virtualization techniques provide an alternative connection via public IPs. A Virtual Private Network (VPN) is a secure way to build an overlay network over Internet and to make a secure connection between private and pubic clouds. For this purpose, the well-known cloud providers offer VPNs like Amazon VPC,[17] Google VPN,[18] and Microsoft Azure VNet[19] . Since we implement a hybrid cloud over Azure and OpenStack clouds, Azure VPN can be used for the hybrid cloud connection.

Microsoft Azure offers three VPN options.[20] Site-to-Site(S2S) meets the requirements of a hybrid cloud construction and needs a VPN gateway with an assigned Public IP over the public cloud side. However, we observed that Azure VPN gateway connection stops after about two and a half hours if remains inactive. Thus, it requires a mechanism to wake up regularly with the help of third-party scripts or built-in Azure mechanisms. In addition, Azure charges users based on the amount of time that the VPN gateway is provisioned. This charge is applied even if no data is sent and received through VPN.

To avoid these issues, we deploy WireGuard as Linux kernel-based VPN recently released. It is designed to be easy to use while providing a robust and secure connection between parties over a network interface encrypted with public key authentication. WireGuard, in contrast to Azure VPN, gives a virtual interface, for example wg0, which can be managed using the standard ip(8) and ifconfig(8) utilities. After configuring the interface wg0 with a private key and public keys of peers with whom it will connect securely, the tunnel between peers becomes operational. This makes WireGuard easier to use and simple compared to IPSec used by Azure VPN gateway. Furthermore, there is no need of VPN gateway and a public-facing IP address in on-premise infrastructure. Fig. 2 illustrates the deployment of Azure VPN and Wiregurad to build a hybrid cloud.

The exploitation of WireGuard instead of Azure VPN brings the following benefits. *Security*: Wireguard authenticates peers through exchanging public keys and encrypts data independent of any vendor/organization. While Azure VPN encrypts data which is under control of a vendor. *Reliability in connection*: WireGuard makes a persistent connection between clouds even without sending and receiving data between two VMs as conducted experiments for 72 h; while Azure VPN does not. *Cost:* WireGuard offers zero-cost services while Azure VPN charges users. *Ping Time and Throughput*: WireGuard outperforms IPSec protocol (Dhall et al., 2012) (exploited by Azure VPN) in both ping time (i.e., response time) and throughput (Donenfeld, 2018). *Interportability*: WireGuard easily provides interportability between any cloud provider through creating Wireguard configuration file. Thus, we use WireGuard to implement the hybrid cloud.

Deployment of Infrastructure Resources: There are several ways to define, preview, and deploy cloud infrastructure resources. Public cloud providers support several embedded tools like command-line interface (CLI), Power Shell, and Portal web services that enable the management of limited resources with static properties. To break these limitations, Infrastructure as Code (IAC) like Terraform[21] support resource deployment and configuration in a dynamic way. Terraform is strictly declarative language and provides a readable and clear syntax for software developers and makes it more desirable to be used in the configuration of infrastructure resources. Hence, we select it to implement resource pooling across OpenStack and Microsoft Azure.

Phases of Hybrid Cloud Implementation: We discuss step-by-step the implementation of hybrid cloud over OpenStack and Microsoft Azure (HybOPAZ) based on on-demand usage model using WireGuard and Terraform. Fig. 3 shows the phases of HybOPAZ implementation

[15] To implement two other usage models, it is required to add two key components on top of the on-demand usage model. The first component is to implement a database to include used and free resources across all cloud datacenters involved in the hybrid cloud. This database allows to make queries to ensure the availability of required resources. The second component is to add a mechanism to verify authentication between cloud datacenters.

[16] A secure connection makes encrypted data by security protocols to guarantee the security of data flow between VMs locating across private and public clouds.

[17] Amazon VPN:https://aws.amazon.com/vpc/.

[18] Google VPN:https://cloud.google.com/vpc/.

[19] Azure VPN:https://azure.microsoft.com/en-au/services/vpn-gateway/.

[20] Point-to-Point (P2P) creates a secure connection from an individual VM in the private cloud to the Azure VNet, while VNet-to-VNet (V2V) makes a secure connection between two VNets in the same or different regions in Azure cloud.

[21] Terraform:https://www.terraform.io.

in which initially a consumer broker network is created within OpenStack (Phase 1). To expand workload to the public cloud in the case of workload spike, we need to create a donor broker network on Azure and connect it to the consumer broker through WireGuard (Phases 2–3). Based on the requirements, we enable HybOPAZ to expand shared networks/subnetworks (phase 4) that can be peered (phase 5) to the local networks. Finally, Phase 6 enables data routing across broker, local and shared networks for HybOPAZ. We discuss the implementation of phases below.

Step 1: This step consists of phases 1–3 of HybOPAZ implementation including creation of consumer and donor broker VMs, and WireGurd configuration between them. As depicted in Fig. 5, we deployed a VM manger in OpenStack to control and store private and public keys required for a secure connection between broker VMs. We wrote several Terraform and shell scripts for the following modules (see Fig. 4) to automate resource sharing between two clouds.

(i) *Donor configuration preparation*: This module creates public and private keys via key generation mechanism provided by WireGuard and saves them in the VM manager. Based on these keys, we create WireGuard configuration file wg0 for the donor broker VM in Azure.

(ii) *Donor broker VM Creation*: VM manager runs the automation scripts to create a virtual network including the donor broker VM. Then, the terraform script installs Wireguard on it, uploads wg0 to it, and starts Wireguard services on it.

(iii) *Consumer Configuration Preparation*: This module performs the same task as for the donor broker VM in module (i). Based on the generated keys and donor broker VM's public key and public IP, we create configuration file for the consumer broker VM in OpenStack.

(iv) *Consumer broker VM creation*: Like module(ii), VM manger creates the consumer broker VM in OpenStack, then installs Wireguard on it, and finally uploads WireGuard configuration file, created in module(iii), to it. In contrast to the donor broker VM, the consumer broker VM cannot start Wireguard in this step because it is not yet authorized to access the donor broker VM.

(v) *Consumer broker VM authorization*: This module allows the consumer broker VM to connect to the donor broker VM through adding the pubic key of the consumer broker (created in module(i)) and the range of IP addresses allowed inside the tunnel (consumer_VPN_IP/32). This authorization functionality is added to Wireguard configuration file in the donor broker VM.

Step 2: This step includes phases 4–6 in the case of workload spikes. In phase 4, the implementation of shared and broker networks is identical in OpenStack. In contrast, this phase is different in Azure since the connection between two subnets with different address spaces via the virtual network peering is provided. Phase 5 connects the shared networks to the broker networks via adding the network interface of shared network (Fig. 5) in OpenStack and virtual network peering in Azure. Phase 6 implements data routing in OpenStack by adding static rules to both routers in the broker and shared networks. Differently, in Azure, the shared network is initially associated with the routing table of the broker network, and accordingly the routing table is modified with static rules.

5. Distributed databases and workload setup

This section initially discusses the selected databases to deploy in the hybrid cloud. Then, it describes the workload for evaluation of these databases.

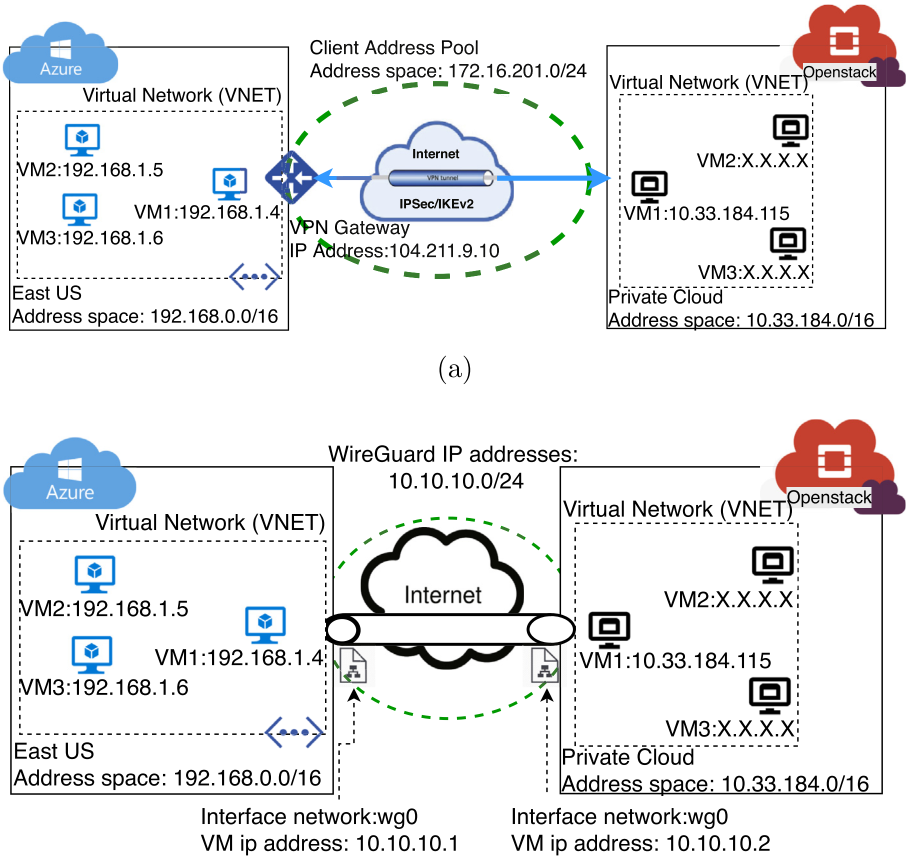


Fig. 2. Connection between OpenStack and Azure using (a) WireGuard and (b) Azure VPN Gateway.

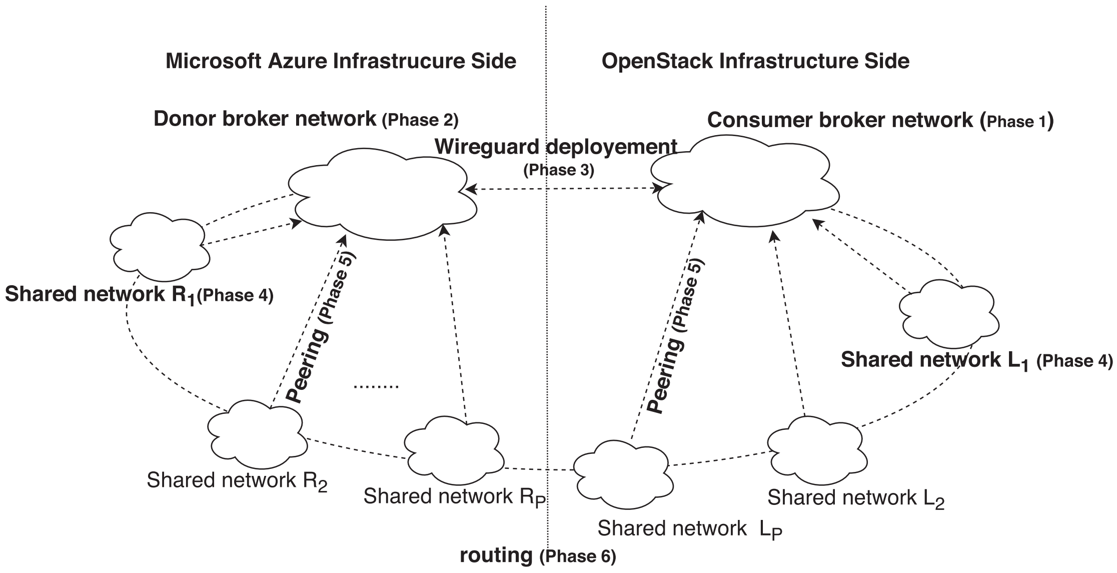
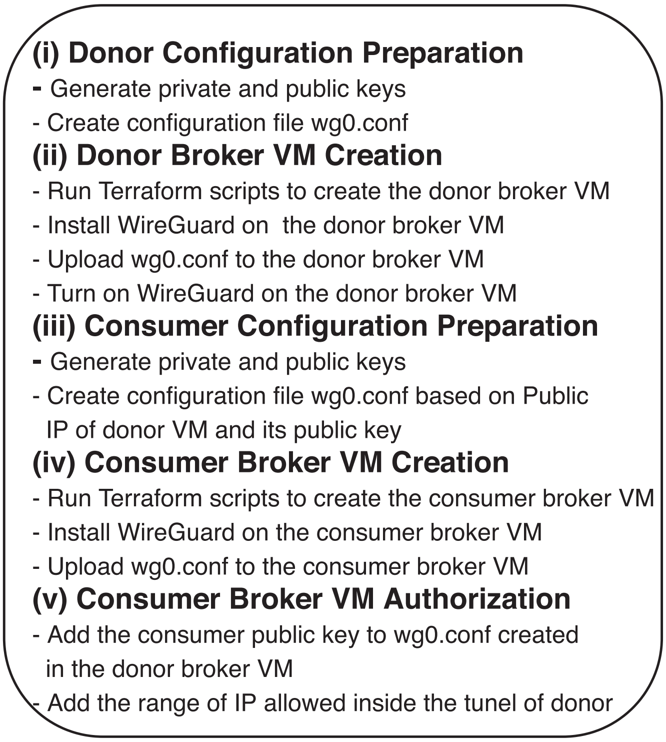

Microsoft Azure Infrastrucure Side
OpenStack Infrastructure Side
Donor broker network (Phase 2)
Consumer broker network (Phase 1)
Wireguard deployement
(Phase 3)
Shared network R1(Phase 4)
Peering (Phase 5)
....
Peering (Phase 5)
Shared network L1 (Phase 4)
Shared network R2
Shared network RP
Shared network LP
Shared network L2
routing (Phase 6)

Fig. 3. Different phases of HybOPAZ implementation.



(i) Donor Configuration Preparation

- Generate private and public keys
- Create configuration file wg0.conf

(ii) Donor Broker VM Creation

- Run Terraform scripts to create the donor broker VM
- Install WireGuard on the donor broker VM
- Upload wg0.conf to the donor broker VM
- Turn on WireGuard on the donor broker VM

(iii) Consumer Configuration Preparation

- Generate private and public keys
- Create configuration file wg0.conf based on Public IP of donor VM and its public key

(iv) Consumer Broker VM Creation

- Run Terraform scripts to create the consumer broker VM
- Install WireGuard on the consumer broker VM
- Upload wg0.conf to the consumer broker VM

(v) Consumer Broker VM Authorization

- Add the consumer public key to wg0.conf created in the donor broker VM
- Add the range of IP allowed inside the tunel of donor

Fig. 4. A procedural step-by-step of donor and consumer broker VMs Configuration preparation, creation, and authorization.

5.1. Distributed databases

The main purpose of the automated implementation of hybrid cloud is to evaluate the performance impact of cloud bursting on distributed database systems. This provides an overview of the performance impact of network bottleneck between two datacenter clouds on different distributed database systems. The criteria selection for the NoSQL databases is maturity and high usage in industry. We also selected MySQL as the leading relational database due to its well-developed and globally exploitation in industry.

MongoDB: MongoDB is a document-oriented database without single point of failure since in case master replica goes down then the secondary replica is selected as a master. MongoDB supports asynchronous master-slave replication, meaning that writes are only served by the master replica and reads by any replica.

Cassandra: Apache Cassandra provides a highly available data services with no single point of failure implying all nodes are equal- no concept of master and slave nodes. In Cassandra, each server can handle read and write with different levels of consistency, where “ONE” is a default level.

Riak: Riak is a document-oriented database system and supports master-less replication architecture without single-point of failure. Riak offers tunable consistency level. By default, Riak supports peer-to-peer replication and eventual consistency.

CouchDB: Couchdb is a document-oriented database and has asynchronous master-slave replication and provides local quorum -based consistency. Each database in couchdb is spilt to 8 shards, while other databases leverage a dynamic sharding policy.

Redis: Redis is an in-memory database and supports asynchronous master-slave replication architecture in which each master can handle several slaves and a slave can act as master to other slaves. In fact, Redis supports non-blocking replication on both master- and slave-side, implying that master can serve queries while slaves are synchronizing with master and slave can handle requests through old version of data.

MySQL Cluster: MySQL Cluster is a distributed and relational database with full SQL support and ACID properties. It provides shard-nothing clustering and auto-sharding for MySQL database. MySQL Cluster exploits synchronous replication via a two-phase commit to guarantee ACID features. MySQL Cluster implemented through Network Database (NDB) is not secure in communication between data nodes in the cluster. Thus, MySQL cluster improves network speed.

We run all these databases based on the default settings as summarized in Table 4. As can be seen, some of those databases do not support data encryption for “data at rest” and “data at motion” and can take benefit from Wiregurad to encrypt data.

5.2. Workload setup

To evaluate cloud bursting at the database level, we leverage YCSB (Cooper et al., 2010) workload that is suitable for store and retrieve data from distributed databases. YCSB consists of *client component* that generates workload and *core workload* that is a set of queries to be executed by the client component. This workload allows to define a config-

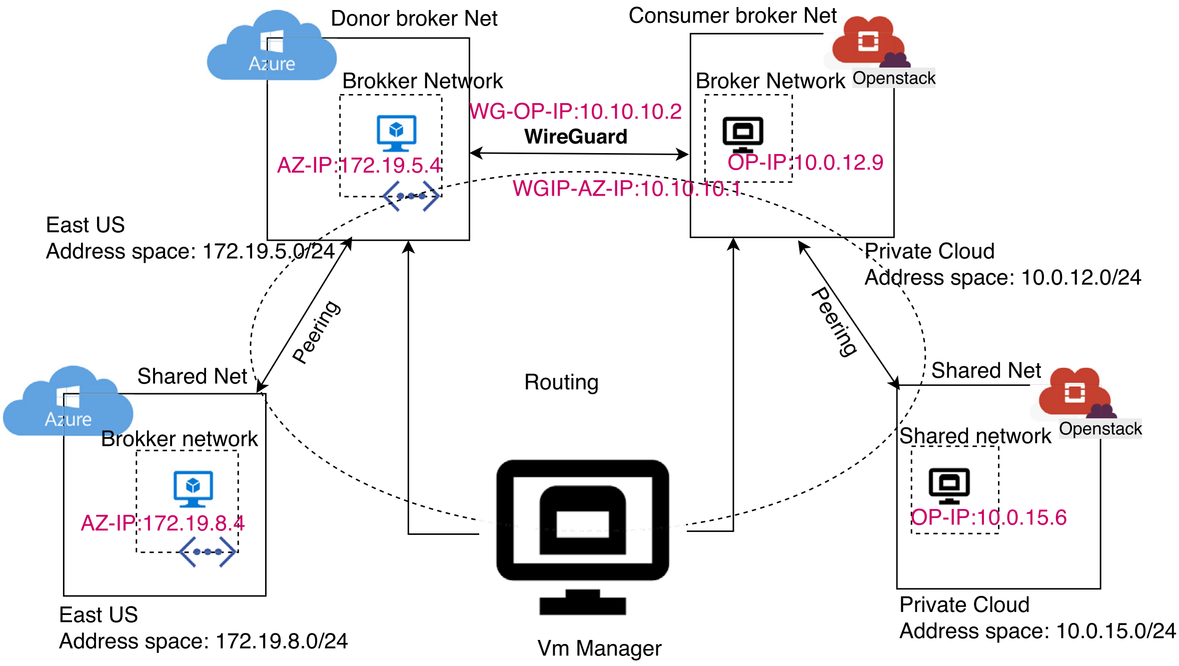


Fig. 5. Implementation of HybOPAZ using WireGuard.

Table 3
Core workload in YCSB.

Workload type	Operations	Label
Workload A	50% Read +50% Update	Read-intensive
Workload B	95% Read +5% Update	Write-intensive
Workload C	100% Read	Read-only
Workload D	95% Read +5% Insert	Read-latest
Workload E	95% Scan +5% Insert	Scan
Workload F	50% Read +50% update	Read-Modify-Write (RMW)

urable distribution of CRUD (create, read, update and delete) operations on data set. Each record in the data set consists of 10 fields, each with a length of 8 bytes. Thus, a record in the data set is 80 byte.

As shown in Table 3, YCSB workload consists of one or two atomic operations and includes three phases: *flush*, *load*, and *run*. Flush phase allows to make clear database from the data already stored. Load phase generates 10000 records of random data and writes in the database. Run phase executes 1000 atomic operations on the written data in database. All these phases run on the local OpenStack to keep validation of on-demand usage model of hybrid cloud.

6. Performance evaluation

We delineate the details of hybrid cloud infrastructure resources and discusses experimental evaluation and practical findings of cloud bursting at the database level.

6.1. Hybrid cloud configuration

We used two clusters for our evaluation based on the HybOPAZ architecture. One group of nodes is deployed in local OpenStack (version:2.3.1) and consists of VM instances with 1 core CPUs, 2 GiB of RAM, and 10 GB disk. Other group of nodes is created in Azure cloud datacenter with Standard_B1ms instances (1 vCPU, 2 GiB RAM, and 4 GiB SSD storage. In all experiments,[22] we deployed 8 total nodes in both groups so that *n* nodes are exploited in the local OpenStack, and *8-n* nodes are burst into the Microsoft Azure cloud in East US region. In our experiments, we considered all permutation of 8 nodes that can be burst into the public cloud. Thus, we have setups of (8,0), (7,1), (6,2), …, (2,6) and (1,7) across both clusters, where the first and second member of each pair is the number of VMs, respectively, deployed in OpenStack and Azure.

We installed databases and configured them on both groups as a single cluster based on the default settings as summarized in Table 4. For this purpose, we deployed two phase scripts: installation and cluster configuration. For MySQL, cluster configuration is slightly different in comparison with NoSQL databases since it requires three types of nodes: *Cluster manager node*, *data nodes*, and *MySQL Server node*. We run cluster manager node and MySQL server node on the same VM instance in OpenStack, and data nodes across both clusters.

With the help of Terraform, we automated deployment and destruction of infrastructure resources across OpenStack and Microsoft Azure. We also leverage Terraform to run configuration files in order to install and build a cluster of distributed databases across the exploited infrastructure. Such automation of implementation allows us to consistently recreate the clusters of distributed database nodes based on the required VMs number, VMs flavor, VMs region, network features, etc.

6.2. Experimental results

In this section, we answer to this research question: *How effective is the Wide Area Network (WAN) cloud bursting at the database level?*. To this end, we initially evaluate HybOPAz in terms of latency and bandwidth between different sub-networks to better understand the limitations of distributed databases deployment across hybrid clouds. Then, we report the results of such deployment in respect to throughput, latency of read

[22] The key purpose of these experiments with such hardware resources is to compare the performance of these databases in the same infrastructure resources. However, we plan to investigate vertical scalability (increasing capacity of hardware) and horizontal scalability (adding more VMs to the pool of resources) for distributed databases across hybrid clouds.

Table 4
Default setting for six distributed databases.

Database	Replica number	Consistency	Data at rest	Data at motion	Version
MongoDB	Full	Eventual	Yes	Yes	3.4
Cassandra	3	Quorum	Yes	No	3.11
Redis	Full	Eventual	No	No	5.0
Riak	3	Eventual	No	Yes	2.2.3
CouchDB	3	Local Quorum	No	Yes	2.2.0
MySQL	2	Strong	Yes	No	7.6.6

Columns “data ate rest” and “data at motion” indicate whether or not a database system respectively supports data encryption at storage level and network level between storage nodes.

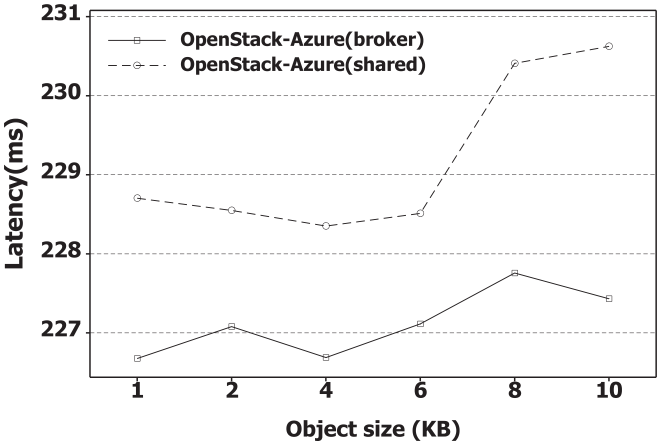


Fig. 6. Latency between broker and shared sub-networks in OpenStack and Azure.

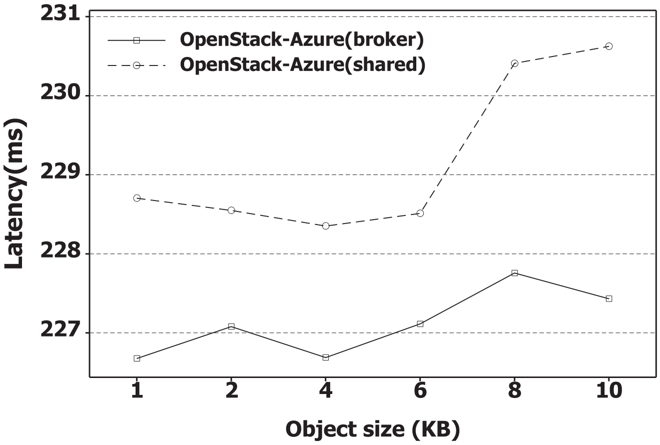


Fig. 7. Latency between two brokers and two shared sub-networks across OpenStack and Azure.

and write operations, and the error percentage for operations performed on distributed databases.

As shown in Fig. 6, latency between broker and shared sub-networks varies between 0.5 and 0.7 ms in OpenStack and 1.1–1.4 ms in Azure to transfer 1000 data packets. In contrast, as depicted in Fig. 7, these values respectively increase by 220 ms–230 ms and 228.5 ms–230.5 ms for two broker and two shared networks in HybOPAZ. Thus, as expected, the latency between two sub-networks across clouds is significantly high and unpredictable. This is confirmed by our experiment, with the help of MTR utility.[23] MTR shows that transfer of data packets between the two shared networks requires to pass 24 hops; 12 of which are in Australia and require 12 ms to pass data packets, with the remaining hops outside of Australia and requiring 218 ms to transfer data packets.

Furthermore, we measured the bandwidth between two shared networks in HynOPAZ through running IPerf3[24] for 10 min. Results show that the bandwidth for download and upload data between these two networks respectively reaches 11191 and 1009 Kbyte/sec. This demonstrates another restriction for transferring data between two clouds. In respect to these limitations, we evaluate cloud bursting at the database level.

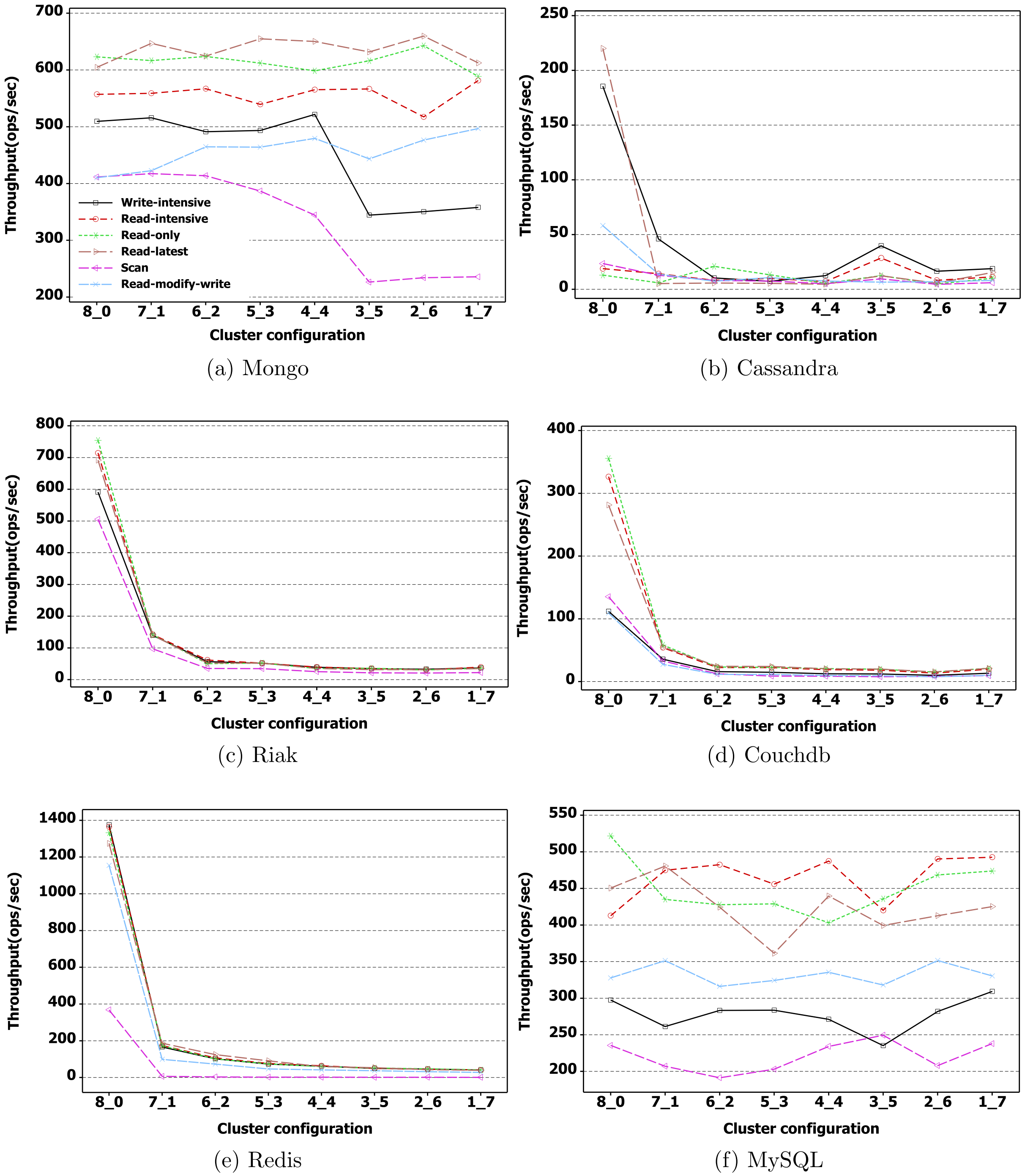
Fig. 8 illustrates the throughput of the six distributed databases against cluster configuration labelled with pairs of (*n*_*m*) in which *n* is the number of VM instances exploited in the local OpenStack and *m* is the number of VM instances burst into Azure cloud datacenter. For each database and cluster configuration, we used a freshly installed and established database cluster and loaded the data. We refer to cluster configuration with pairs (8_0) and (1_7) as *non-bursting* and *full-bursting* respectively. All pairs, except (8_0), are referred to as hybrid cluster configurations. It should be noted that in the full-bursting setting, we still exploit one VM instance in OpenStack due to keeping the definition of hybrid cloud.

As depicted in Fig. 7, in the experiment with the non-bursting setting, MongoDB achieves the highest throughput between 550 and 630 ops/sec for read-related (i.e., read-only, read-latest and read-intensive) workloads. This value reduces for the write-intensive workload (500 ops/sec), followed by scan and read-modify-write (400 ops/sec) workloads. As the cluster configuration changes from non-bursting to bursting (i.e., cluster configurations of (7_1), (6_2), …, (1_7)), the throughput of MongoDB slightly improves for read-related workloads, and increases by 10% for the read-modify-write workload. By contrary, for write-intensive and scan workloads, the throughput of MongoDB significantly reduces when more than half of the resources are provided through the public cloud. In precise, this value declines from 500 to 350 ops/sec for write-intensive and from 350 to 230 ops/sec for scan when cluster configuration changes from (4_4) to (3_5).

As illustrated in Fig. 8b, Cassandra reaches 180–230 ops/sec for write- and read-latest workloads in non-bursting. For other workloads, these values decreases to 25–55 ops/sec. When Cassandra bursts into the public cloud, the throughput for write-intensive and read-latest workloads declines significantly, while its effect on other workloads is less. With the increment in the number of VM instances bursting into the public cloud, Cassandra’s throughput diminishes in cluster configurations of (5_3) and (2_6) and increases in cluster configurations of (6_2) and (3_5).

Fig. 8c and d represent the throughput for Riak and Couchdb databases. For non-bursting, Riak has a throughput of 500–750 ops/sec, while Couchdb has a throughput of 280–350 ops/sec for read-related workloads, and 100–140 ops/sec for the remaining workloads. Cloud bursting leads to 50–100 ops/sec and less 50 ops/sec for Riak and Couchdb respectively. As can be seen, this performance is roughly con-

[23] MTR:https://www.bitwizard.nl/mtr/.
[24] iPerf3: https://iperf.fr/.



(a) Mongo

(b) Cassandra

(c) Riak

(d) Couchdb

(e) Redis

(f) MySQL

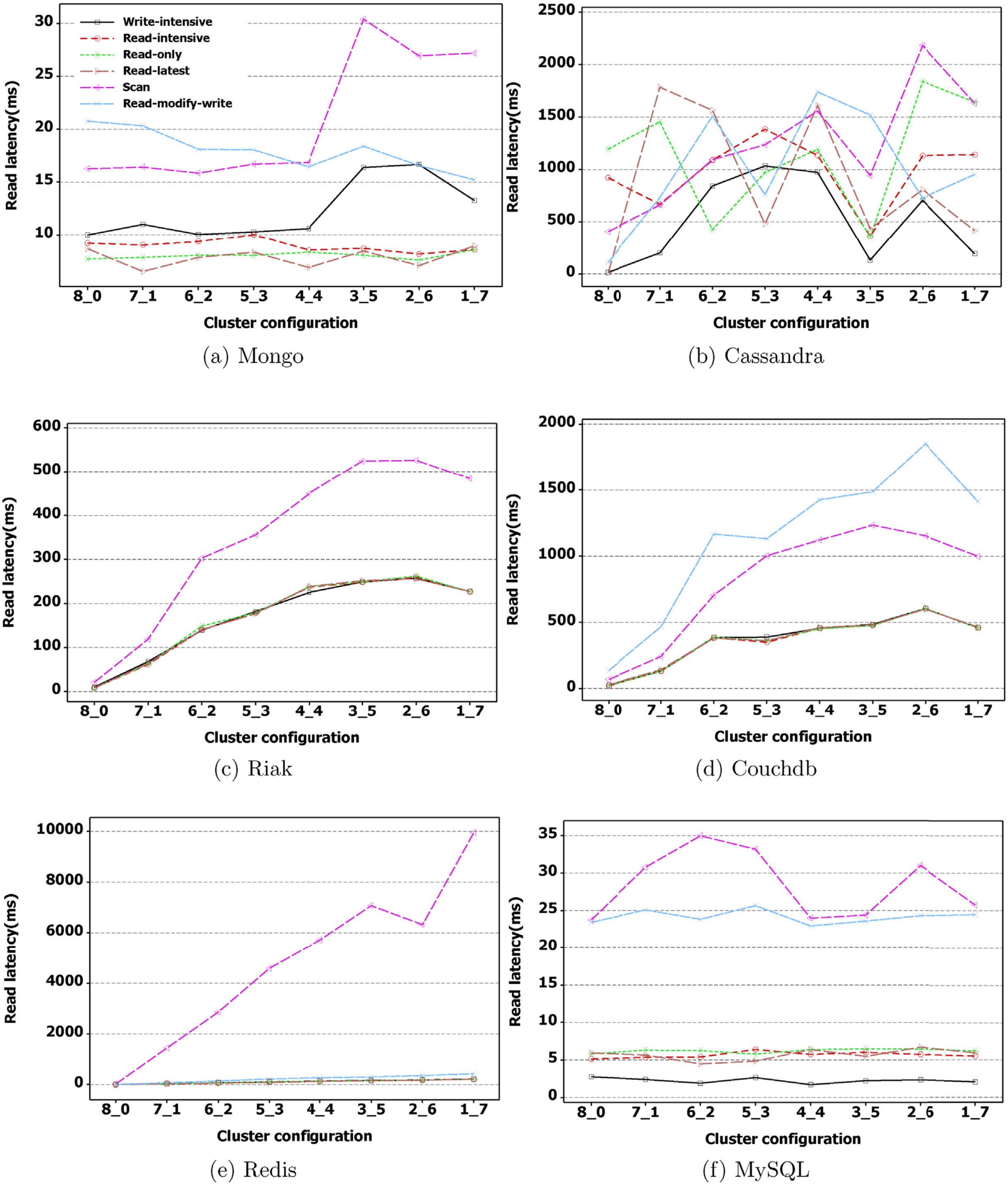
Fig. 8. Throughput for distributed databases. Value *n_m* in axis X represents *n* nodes in the private cloud and *m* nodes in the public cloud are deployed.

stant for all workloads with different hybrid cluster configurations.

Fig. 8e depicts the throughput for Redis databases. In non-bursting, Redis has the best performance in comparison to all databases due to being memory-based. Thus, Redis is not optimized to work well under network-intensive workloads. The results show that cloud bursting is still effective if Redis is provisioned with public cloud for less than half of its required resources.

The throughput of MySQL is shown in Fig. 8f. In the non-bursting setting, MySQL attains the best throughput for read-related workloads (420–530 ops/sec) and the worst one for scan (280 ops/sec) and write-intensive (300 ops/sec) workloads. Similar to MongoDB, MySQL's throughput increases for read-intensive and read-modify-write workloads as cluster configuration changes from non-bursting to bursting (i.e., different hybrid cluster configurations). Like MongoDB, the throughput of MySQL for read-related workloads is better than for other workloads when cluster configuration varies from (7_1) to (1_7).

Discussion: From the results discussed above, we make the following remarks. Under the conditions of the conducted experiments, MongoDB exhibits the best performance in throughput for different workloads for all hybrid cluster configurations. This performance is closely followed by MySQL with a reduction of 16%–28% for read-related workloads and a decrease of 34%–43% for write-intensive, scan, and read-modify-write workloads as the cluster configuration changes from (8_0) to (4_4). The high latency degrades the performance of Cassan-



(a) Mongo

(b) Cassandra

(c) Riak

(d) Couchdb

(e) Redis

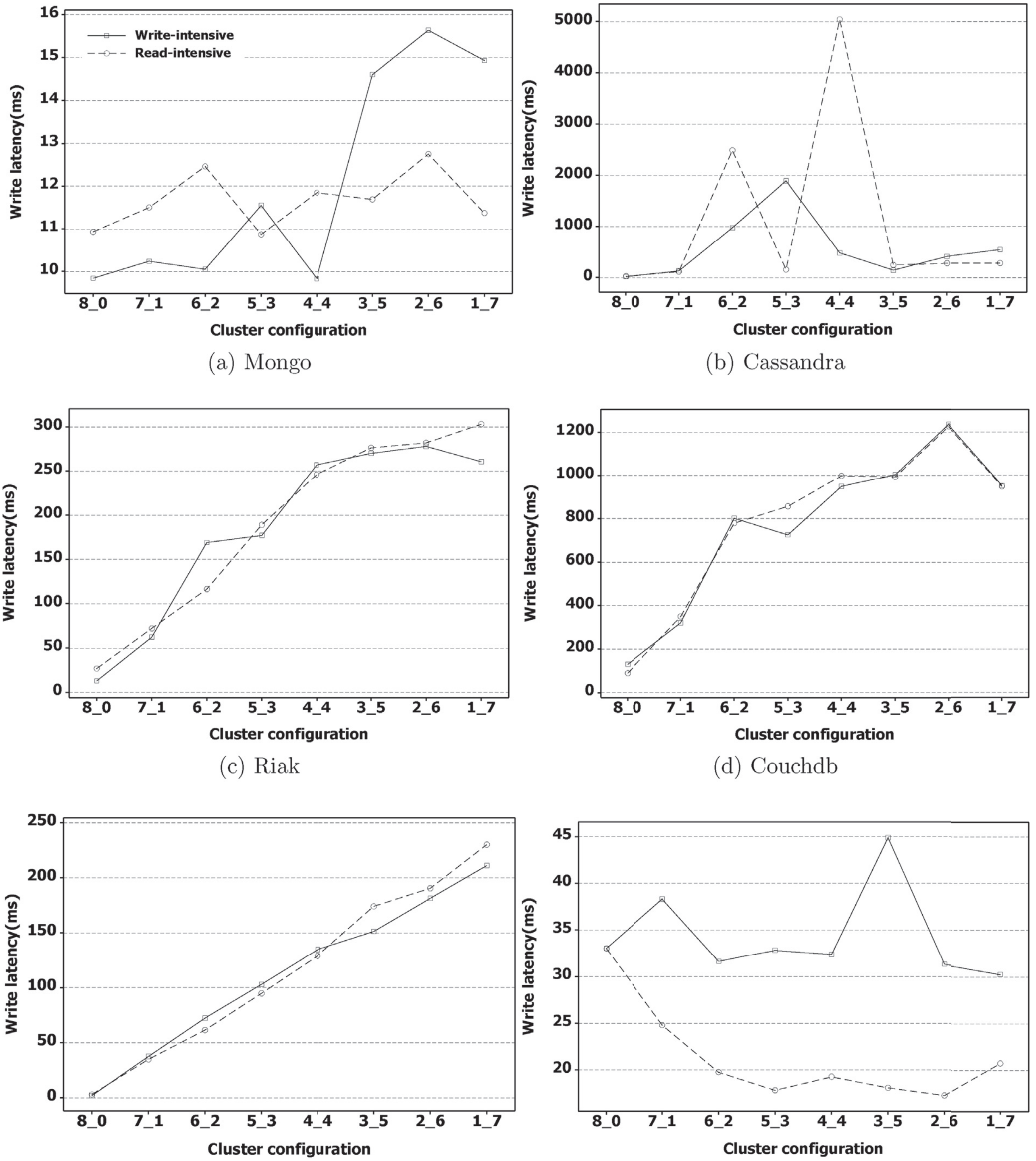
(f) MySQL

Fig. 9. Read latency for distributed databases. Values *n_m* in axis *X* represent *n* nodes in the private cloud and *m* nodes in the public cloud.

dra, Riak, and Couchdb due to using the quorum-based technique for read and write commit operations. Redis has also the same performance degeneration because it is not designed to be deployed across WAN.

In Fig. 9, the read latency can be seen. The read latency in MongoDB with non-bursting is less than the one for the hybrid cluster configurations (i.e., (7_1) - (1_7)) for all workloads except scan and write-intensive. The deployment of MongoDB with the hybrid cluster configurations, the read latency reduces by 25% for the read-modify-write workload. By contrast, for write-intensive and scan, the read latency substantially increases as the cluster configuration changes from (4_4) to (3_5), followed by a reduction of 5% for the cluster configuration of (1_7).

Fig. 9b illustrates the read latency of Cassandra. The read latency increases as the setting changes to the hybrid cluster configurations, which is consistent with its low throughput. This database has unstable behaviour in read latency due to reading data from remote datacenter based on the quorum-based technique. One interesting observation of the results is that Cassandra achieves the lowest read latency in cluster configurations of (5_3) and (3_5) among all the hybrid cluster configurations. This might be because Cassandra makes a read commitment



(a) Mongo

(b) Cassandra

(c) Riak

(d) Couchdb

(e) Redis

(f) MySQL

Fig. 10. Write latency for distributed databases. Values *n_m* in axis *X* represent *n* nodes in the private cloud and *m* nodes in the public cloud.

based on the quorum-based technique from three replicas as default setting.

As shown in Fig. 9c, Riak reduces the read latency more than three times in comparison to the Cassandra for the cluster configurations of (7_1)-(2_6). Couchdb is comparable in the read latency with Cassandra although it is more stable due to exploiting the *local* quorum-based technique to commit operations (9d). Fig. 9e illustrates the read latency for Redis. As expected, the Redis's read latency for all workloads (except scan) is the lowest in comparison to ones for other databases because Redis is an in-memory database. However, this database yields high read latency for scan workload as more nodes are exploited in the public cloud.

Like MongoDB, MySQL achieves low latency for read operations: about 2.5 ms for the write-intensive workload and 5 ms for read-related workloads. These values roughly remain constant for all hybrid cluster configurations (Fig. 9f). This is because MySQL performs well in pairing replica nodes across clouds. However, MySQL imposes higher latency for scan (22 ms–25 ms) and read-modify-write (22 ms–35 ms). It should be noted that the read latency for the scan workload reduces when the number of nodes, bursting into the public cloud, is a multiple of 2.

Fig. 10 shows the write latency for the write- and read-intensive workloads including 50% and 5% write operations respectively. Among NoSQL databases, MongoDB has the lowest write latency varying from 10 ms to 15 ms. In this performance metric, Redis comes in the second

Table 5
Percentage of error for operations operated on distributed databases.

Database name	Write-intensive	Read-intensive	Read-modify-write
MongoDB	11.33%	11.13%	11.46%
Cassandra	20%	25%	26%
Couchdb	3.5%	0.27%	0%
Redis	0.27%	0.63%	0.77%
Riak, MySQL	0%	0%	0%

rank and its write latency increase as the number of Azure VM instances involved in the hybrid cloud raises. Riak has lower write latency compared to Cassandra and Couchdb especially for the hybrid cluster configurations of (7_1)-(2_6). In contrast to all databases, MySQL's write latency decreases as the cluster configuration changes from non-bursting to bursting.

Discussion: The overall results show that MongoDB and MySQL have the lowest read latency, while Cassandra behaves in unstable way for all workloads. In respect to the write latency, MongoDB and MySQL again produce the lowest write latency although in some hybrid cluster configurations they have spikes. These spikes might be resulted from the network latency fluctuations, as shown in Fig. 7.

Table 5 summarize the error percentage for operations performed on distributed databases in the case of hybrid cluster configurations. Error for one operation means that for some reasons (e.g., network partitioning or consistency violation) the YCSB client is not able to perform or verify an operation. Results show that Riak and MySQL are error-free in operations conducted by the YCSB client. In contrast, Cassandra has the highest error percentage for operations (20%–26%), followed by MongoDB. These results are consistent with attaining the highest throughput by MongoDB deployment in the hybrid cloud. Surprisingly, although Cassandra has the highest percentage in error for operations, it could not outperform MongoDB in throughput. This might because MongoDB supports eventual consistency while Cassandra offers quorum-based consistency by default. It is worth nothing that in the non-bursting setting, all databases 5 are operating error-free.

6.3. Practical observations

We report additional observations and findings from a practitioner's perspective during the implementation of hybrid cloud and deployment of the six distributed databases.

To make a secure connection between the local OpenStack and Microsoft Azure clouds, we tested both default Microsoft Azure VPN and WireGurd. The deployment of WireGuard is more simple because we only need to exchange private and public keys of the VMs involved. While in its counterpart, we need a public IP address in Microsoft Azure for VPN gateway to configure many settings in the OpenStack side to establish connection. In addition, WireGuard is free and provides a reliable connection. In contrast, Microsoft Azure is costly and depends on the amounts of data transferred. Another advantage of WireGuard is the high flexibility and universal applicability for other cloud providers such as AWS and Google. In short, due to these features, we selected WireGuard to make a secure connection.

There are several ways to deploy virtualized cloud infrastructure. As an example, Microsoft Azure supports a number of tools such as Azure command-line interface (CLI), Azure Power Shell, and Azure Portal.[25] However, these tools are appropriate when a limited number of resources need to be managed. In efforts to improve automation of virtualized resource management several solutions such as Terraform and Ansible have been developed. We selected Terraform due to high adoption, syntax simplicity, and multi-cloud support. Terraform enabled strongly reproducible experimentation with high performance. Interestingly, infrastructure deployment in the local OpenStack cloud is significantly faster (up to a factor of 10) compared to Azure deployment.

In respect to the deployment of 6 databases, we found that their installation and cluster configuration is different especially the configuration of NoSQL databases and MySQL. MongoDB and Cassandra databases follow a master-slave architecture for cluster configuration, while Redis, Riak and CouchDB treat all deployed nodes in the same way. In addition, some databases have further specific restrictions such as number of nodes for MySQL or number of master nodes for Cassandra. Similarly, authentication setup required for CouchDB and MySQL complicates unattended installation. These database-specific differences led to the necessity to design flexible and extendable installation scripts architecture. As a result, the developed deployment system is easily expandable to support other new databases.

Lastly, the benchmarking phase was complete using YCSB. Due to individual database differences, a set of database-specific YCSB invocation scripts have been developed. These included variation on authentication details, uri format, initialization queries, port numbers and so on. A notable problem was a lack of compatibility with CouchDB. Thus, patching and recompiling the CouchDB YCSB connector was required.[26] Additional saturation-related consideration raised by Rabl et al. (2012) was not applicable for our experiment scenarios due to a limited bandwidth and high latency between the public and private clouds.

7. Conclusions and future work

In this paper, we presented an automation of hybrid cloud implementation using (i) WireGuard as a Linux-based VPN to make a secure connection between public and private clouds, and (ii) Terraform as a software tool to deploy infrastructure resources based on the required number of VM instances, VM flavour, security group network, subnetwork, and so on. We deployed the implemented hybrid cloud to evaluate cloud-bursting at the level of distributed databases. Our evaluation reveals that MongoDB and MySQL Cluster work well in throughput and latency of read and write operations as if they burst into the public cloud. In contrast, Cassandra, Riak, Couchdb, and Redis, exhibit low performance especially when they supply more than half of their resources via cloud bursting across clouds locating at a long distance from each other. This leads to high and unpredictable latency that negatively affects the performance of these databases. Thus, it is required either to deploy such databases across data centers locating at a short distance (i.e, within a continent), or improve them in data model, data sharding and replication policy. Overall, while experiments revealed that cloud-bursting does not improve the database performance, under certain condition capacity improvement might still be beneficial.

For future work, we plan to investigate the static and on-the-fly dynamic up and down scaling of distributed databases deployment in hybrid clouds. Additionally, we are interested in determining the impact of distance between private and public clouds and replication factor on the throughput and latency of operations in our study. Finally, we plan to extend our tested architecture across different regions of the public cloud deployment and design optimally data placement algorithms to improve performance with the fixed and variable number of VM instances.

CRediT authorship contribution statement

Yaser Mansouri: Formal analysis, Writing - original draft. **Victor Prokhorenko:** Conceptualization, Methodology. **M. Ali Babar:** Project administration, Supervision.

[25] Microsoft Azure: https://azure.microsoft.com/en-au/overview/.

[26] YCSB-CouchDB-Binding: https://github.com/akhildixit/YCSB-CouchDB-Binding.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We thank Faheem Ullah and anonymous reviewers for their valuable comments in improving the quality of the paper. This work was supported in part by the DST Group.

References

Abdi, S., PourKarimi, L., Ahmadi, M., Zargari, F., 2017. Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds. Future Generat. Comput. Syst. 71 (C), 113–128.

Abramova, V., Bernardino, J., 2013. Nosql databases: Mongodb vs cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering, C3S2E '13. ACM, New York, NY, USA, pp. 14–22.

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2010. A view of cloud computing. Commun. ACM 53, 50–58, https://doi.org/10.1145/1721654.1721672.

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generat. Comput. Syst. 25 (6), 599–616.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software Pract. Ex. 41 (1), 23–50, https://doi.org/10.1002/spe.995.

Calheiros, R.N., Vecchiola, C., Karunamoorthy, D., Buyya, R., 2012. The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. Future Generat. Comput. Syst. 28 (6), 861–870 including Special sections SS: Volunteer Computing and Desktop Grids and SS: Mobile Ubiquitous Computing.

Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R., 2010. Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10. ACM, New York, NY, USA, pp. 143–154.

Dhall, H., Dhall, D., Batra, S., Rani, P., 2012. Implementation of ipsec protocol. In: 2012 Second International Conference on Advanced Computing Communication Technologies, pp. 176–181.

Donenfeld, J.A., 2018. Wireguard: next generation kernel network tunnel. In: White Paper, pp. 1–20. https://www.wireguard.com/papers/wireguard.pdf.

Han, Jing, Haihong, E., Le, Guan, Du, Jian, 2011. Survey on nosql database. In: 2011 6th International Conference on Pervasive Computing and Applications, pp. 363–366.

Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., Matser, C., 2015. Performance evaluation of nosql databases: a case study. In: Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems, PABS '15. ACM, New York, NY, USA, pp. 5–10.

Kuhlenkamp, J., Klems, M., Rss, O., 2014. Benchmarking scalability and elasticity of distributed database systems. Proc. VLDB Endow. 7 (12), 1219–1230.

Li, Y., Manoharan, S., 2013. A performance comparison of sql and nosql databases. In: 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), pp. 15–19, https://doi.org/10.1109/PACRIM.2013.6625441.

Li, C., Tang, J., Luo, Y., 2018. Towards operational cost minimization for cloud bursting with deadline constraints in hybrid clouds. Cluster Comput. 21 (4), 2013–2029.

Lima, I., Oliveira, M., Kieckbusch, D., Holanda, M., Walter, M.E.M.T., Arajo, A., Victorino, M., Silva, W.M.C., Lifschitz, S., 2016. An evaluation of data replication for bioinformatics workflows on nosql systems. In: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 896–901.

Loreti, D., Ciampolini, A., 2015. A hybrid cloud infrastructure for big data applications. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, pp. 1713–1718.

Mansouri, Y., Toosi, A.N., Buyya, R., 2017. Data storage management in cloud environments: taxonomy, survey, and future directions. ACM Comput. Surv. 50 (6), 91:1–91:51, https://doi.org/10.1145/3136623.

Moschakis, I.A., Karatza, H.D., 2015. Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing. J. Syst. Software 101 (C), 1–14.

Rabl, T., Gmez-Villamor, S., Sadoghi, M., Munts-Mulero, V., Jacobsen, H.-A., Mankovskii, S., 2012. Solving big data challenges for enterprise application performance management. Proc. VLDB Endow. 5 (12), 1724–1735.

Rimal, B.P., Choi, E., Lumb, I., 2009. A taxonomy and survey of cloud computing systems. In: 2009 Fifth International Joint Conference on INC, IMS and IDC, pp. 44–51.

Toosi, A.N., Sinnott, R., Buyya, R., 2018. Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using aneka. Future Generat. Comput. Syst. 79, 765–775.

Tuli, S., Sandhu, R., Buyya, R., 2020. Shared data-aware dynamic resource provisioning and task scheduling for data intensive applications on hybrid clouds using aneka. Future Generat. Comput. Syst. 106, 595–606.

Vecchiola, C., Calheiros, R.N., Karunamoorthy, D., Buyya, R., 2012. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. Future Generat. Comput. Syst. 28 (1), 58–65.

Xiong, F., Yeliang, C., Lipeng, Z., Bin, H., Song, D., Dong, W., 2016. Deadline based scheduling for data-intensive applications in clouds. J. China Univ. Posts Telecommun. 23 (6), 8–15.

Xu, X., Zhao, X., Ruan, F., Zhang, J., Tian, W., Dou, W., Liu, A., 2017. Data placement for privacy-aware applications over big data in hybrid clouds. Secur. Commun. Network. 2017, 1–15, https://doi.org/10.1155/2017/2376484.

Zhou, J., Wang, T., Cong, P., Lu, P., Wei, T., Chen, M., 2019. Cost and makespan-aware workflow scheduling in hybrid clouds. J. Syst. Architect. 100, 101631.

Yaser Mansouri is a researcher with the Centre for Research on Engineering Software Technologies (CREST) at the University of Adelaide working on a project funded by DST Group. Yaser obtained his Ph.D. from Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, the University of Melbourne, Australia. Yaser was awarded first-class scholarship, International Postgraduate Research Scholarship (IPRS) and Australian Postgraduate Award (APA) supporting his Ph.D. studies. His research interests cover the broad area of Distributed Systems, with special emphasis on data replication and management in cloud storage services.

Victor Prokorenko is a researcher with the Centre for Research on Engineering Software Technologies (CREST) at the University of Adelaide. Victor has more than 14 years of experience in software engineering with main areas of expertise including investigation of technologies related to software resilience, trust management and big data solutions hosted within OpenStack private cloud platform. Victor has obtained a PhD in Computer Science from the University of South Australia.

M. Ali Babar is a Professor in the School of Computer Science, University of Adelaide. He is a honorary visiting professor at the Software Institute, Nanjing University, China. Prof Babar has established an interdisciplinary research centre, CREST - Centre for Research on Engineering Software Technologies, where he leads the research and research training of more than 20 (10 PhD students) members. He leads a theme, Platforms and Architectures for Cybersecurity as Service, of the Cyber Security Cooperative Research Centre (CSCRC). Prof Babar has authored/co-authored more than 200 peer-reviewed publications through premier Software Technology journals and conferences. In the area of Software Engineering education, Prof Babar led the University's effort to redevelop a Bachelor of Engineering (Software) degree that has been accredited by the Australian Computer Society and the Engineers Australia (ACS/EA). He coordinates both undergraduate and postgraduate programs of Software Engineering at the University of Adelaide. Prior to joining the University of Adelaide, he spent almost 7 years in Europe (Ireland, Denmark, and UK) working as a senior researcher and an academic. Before returning to Australia, he was a Reader in Software Engineering with the Lancaster University.