CS565 Cloud Computing [AWS Lab]

1. Purpose of the Lab

Understand definitions of various Amazon Web Services (AWS) and their use in cloud computing based web applications that are accessible over the Internet through an AWS account.

2. References to guide Lab work

- *Chapter 1, Distributed and Cloud Computing,* Hwang, Fox & Dongarra
- *Chapter 1, Programming Amazon EC2, Vliet and Paganelli*
- AWS Free Usage for Education:
    - Overview of AWS, http://media.amazonwebservices.com/AWS_Overview.pdf
    - Services on the free usage tier, http://aws.amazon.com/free/
    - Make the most of your free monthly usage, http://docs.aws.amazon.com/gettingstarted/latest/awsgsg - freetier/TestDriveFreeTier-monthly.html
- AWS Documentation: http://aws.amazon.com/documentation/
- AWS Reference Architectures: http://aws.amazon.com/architecture/
- General AWS Reading: T. Morgan, "A Rare Peek Into The Massive Scale of AWS", Nov. 2014 - http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/
- AWS educational resources (AWS credit, training, learning resources) https://aws.amazon.com/education/awseducate/

3. Lab Steps and output collection guidelines

Figure 1: Lab Steps Overview

The Figure 1 shows the required steps to be followed in order to successfully create an account credited with funds for this course. You will need to understand pricing conditions and services documentation related with 'AWS free usage tier', understand AWS Architecture, create your 'AWS account' and request for $100 credit. Then, you will launch your first AWS EC2 (Elastic Compute Cloud) instance. The final step will show you the ways to connect to your instance through local Linux/Mac or Windows computer.

3.1 Amazon Web Services

Take your time in order to understand the conditions of free usage that involve free account availability, restrictions in terms of instance types, pay-as-you-go service rates, operating systems that are under the free usage condition, and free usage accumulation, detailed information can be found on http://aws.amazon.com/free/.

AWS Free Tier (12 Month Introductory Period):

These free tier offers are only available to new AWS customers, and are available for 12 months following your AWS sign-up date. When your 12 month free usage term expires or if your application use exceeds the tiers, you simply pay standard, pay-as-you-go service rates (see each service page for full pricing details). Restrictions apply; see offer terms for more details.

Elastic Compute Cloud (EC2)

- 750 hours of Amazon EC2 Linux t2.micro instance usage (1 GiB of memory and 32-bit and 64-bit platform support) – enough hours to run continuously each month*
- 750 hours of Amazon EC2 Microsoft Windows Server† t2.micro instance usage (1 GiB of memory and 32-bit and 64-bit platform support) – enough hours to run continuously each month*
- 750 hours of an Elastic Load Balancer plus 15 GB data processing*
- 30 GB of Amazon Elastic Block Storage in any combination of General Purpose (SSD) or Magnetic, plus 2 million I/Os (with EBS Magnetic) and 1 GB of snapshot storage*
- 500 MB-month of Amazon EC2 Container Registry storage for new customers*

Amazon Simple Storage Service (S3)

- 5 GB of Amazon S3 standard storage, 20,000 Get Requests, and 2,000 Put Requests*

Data Transfer

- 15 GB of bandwidth out aggregated across all AWS services*

Amazon Data Pipeline

- 3 low frequency preconditions running on AWS per month*
- 5 low frequency activities running on AWS per month*

Amazon ElastiCache

- 750 hours of Amazon ElastiCache cache.t2micro Node usage - enough hours to run continuously each month.*

Amazon CloudFront

- 50 GB Data Transfer Out, 2,000,000 HTTP and HTTPS Requests of Amazon CloudFront*

Amazon AppStream

- 20 free hours per month*

Go through the http://aws.amazon.com/documentation/ to find detailed information of each service that AWS provides. Pay special attention to the service groups: Getting started with AWS, Compute, Storage & Content Delivery and Database.

3.2. AWS Architecture Center

You will need to understand overall http://aws.amazon.com/architecture/ to help you build your application architecture customized according to your requirements, and for maximizing the AWS services usage. Web application hosting related customization example is shown below in Figure 1.

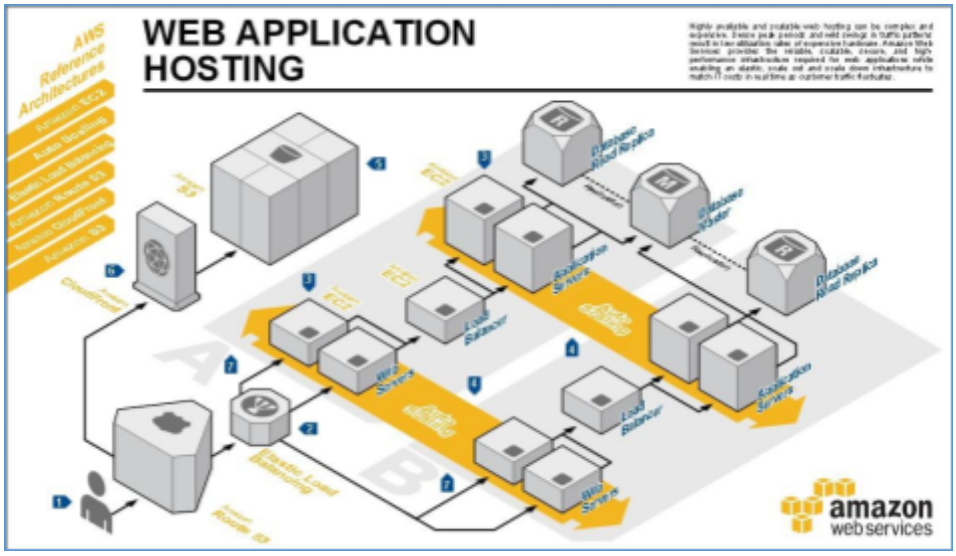


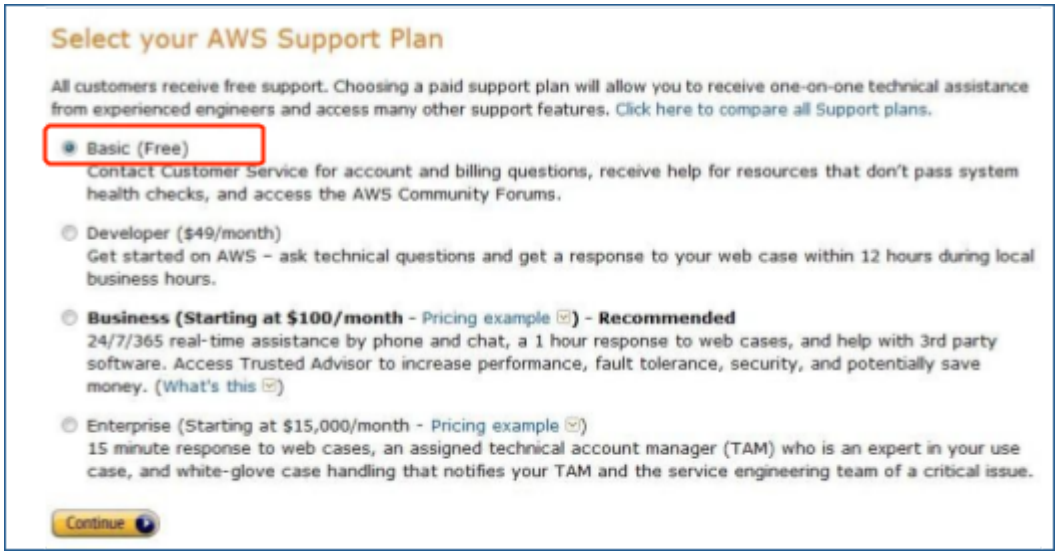
Figure 1: Example application customization of AWS architecture

3.3. AWS Account Creation

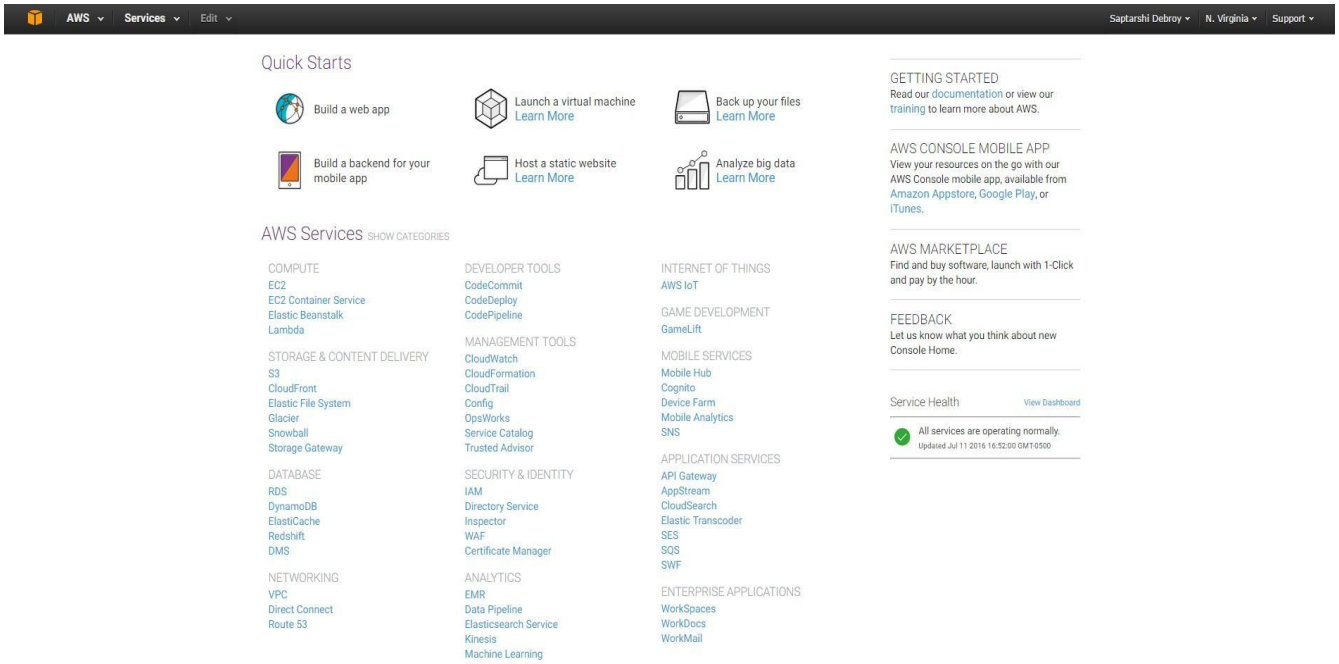
Create an (AWS) Amazon Web Service account in http://aws.amazon.com by clicking the button 'Create a Free Account' and follow the instructions. A credit/debit card and a cellphone/Landline number is required.

- Follow the instruction to create your account. At some point you will also need to enter your credit/debit card information.

- Don't forget to select 'Basic (Free)' Support plan to access to AWS free services (if that information is required).



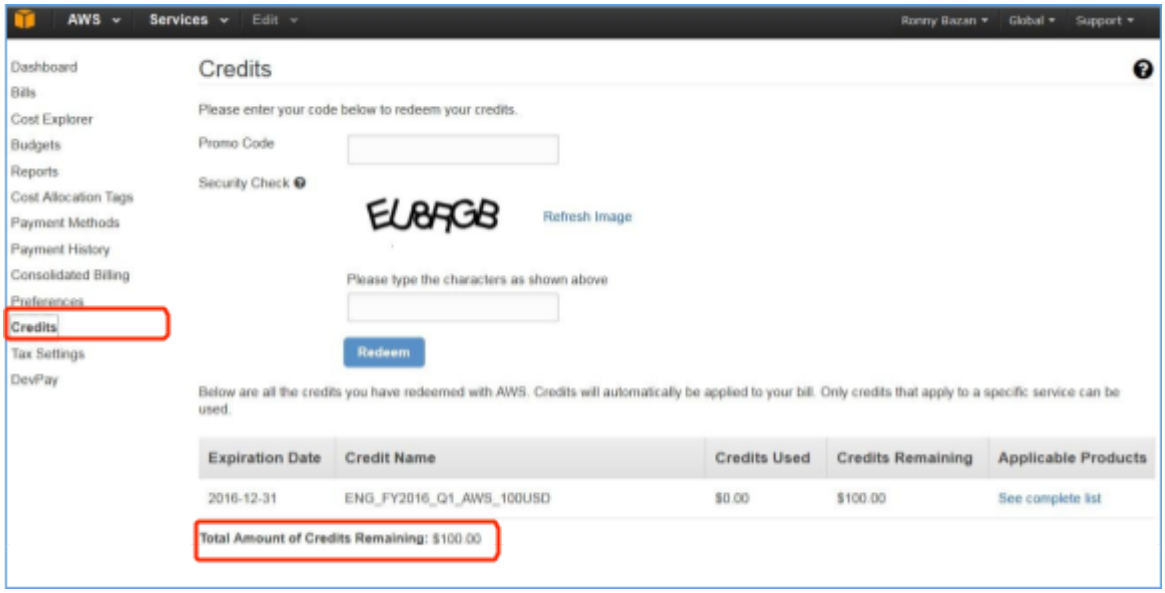
- Once you create your account you will see all AWS services available for you.



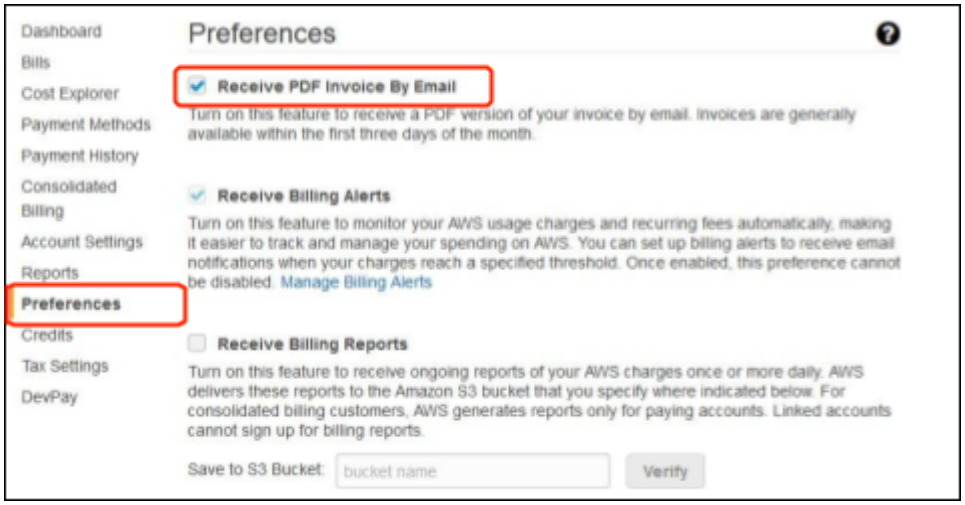
3.4. Add AWS Education Credit for $100 to your Account.

- In https://aws.amazon.com/education/awseducate/apply/ Apply for AWS credit using the 'Apply for AWS Educate for students' link.

- Once you received an email from AWS with the Promo Code, activated it by enter it on ‘Credits’ tag as shown below.



- You will be able to see your detailed usage and Credits Balance by accessing ‘Bills’ option in the left menu.
- Another useful option is to enable ‘Receive PDF Invoice my Email’ as well ‘Receive Billing Alerts’ to keep track of the usage.



3.5 Launching your first AWS Instance

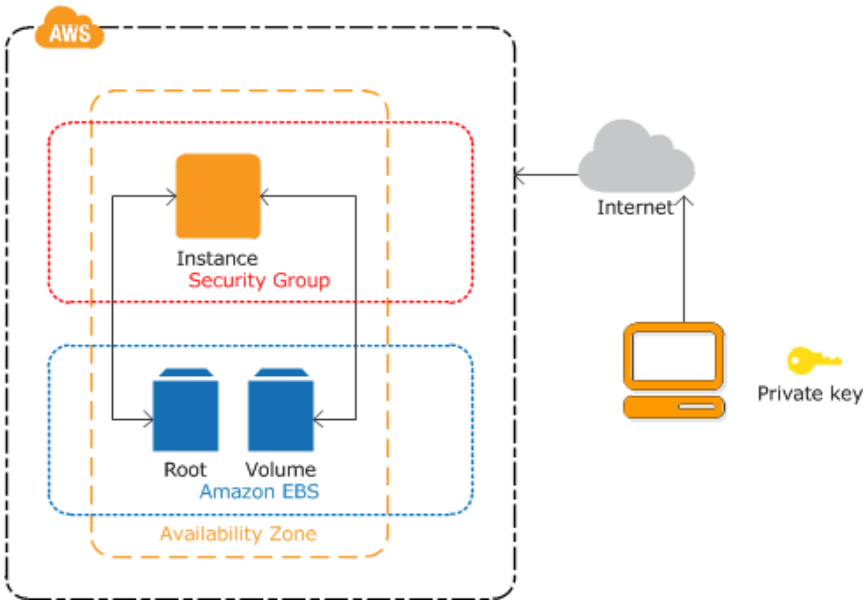
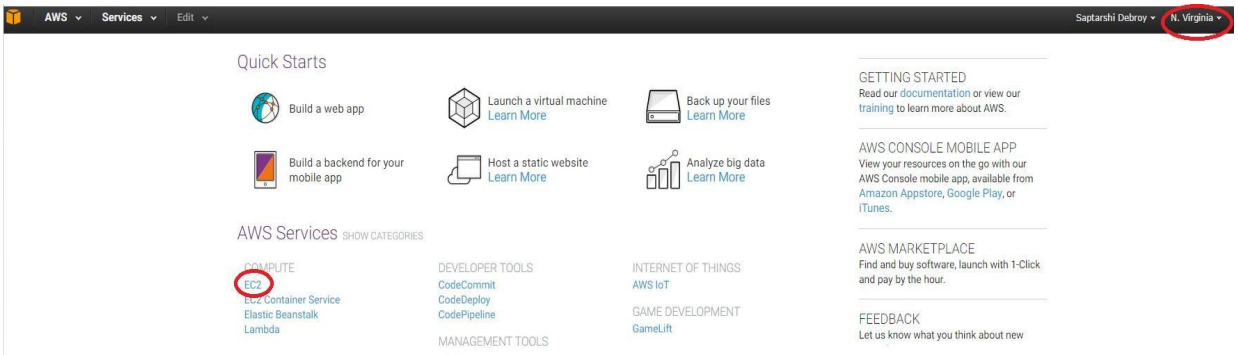


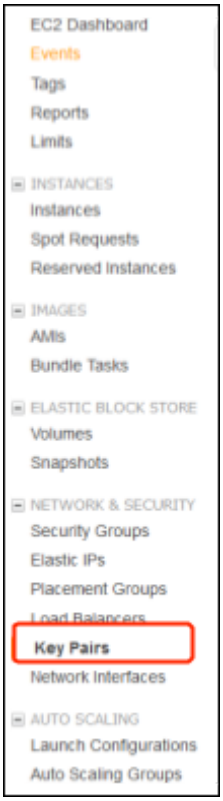
Figure 3: Overview of an AWS Instance

Figure 3 shows the instance architecture to be configured in this Lab. Using your AWS account, you will launch a virtual instance created in a new 'Volume' from an Amazon EBS-backed instance snapshot (called 'Root'), in order to access your reserved infrastructure resources over the Internet; you will need to create key pairs and secure it through a security group; all the infrastructure will be created in a specific zone.

3.5.1 Click 'Console Home', make sure to select the US East (N. Virginia) region in the top-right part of your screen and select AWS EC2 service (Elastic Compute Cloud).



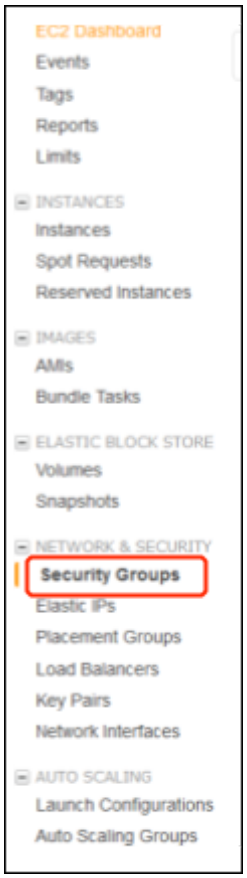
3.5.2 In left menu select “Key Pairs”.



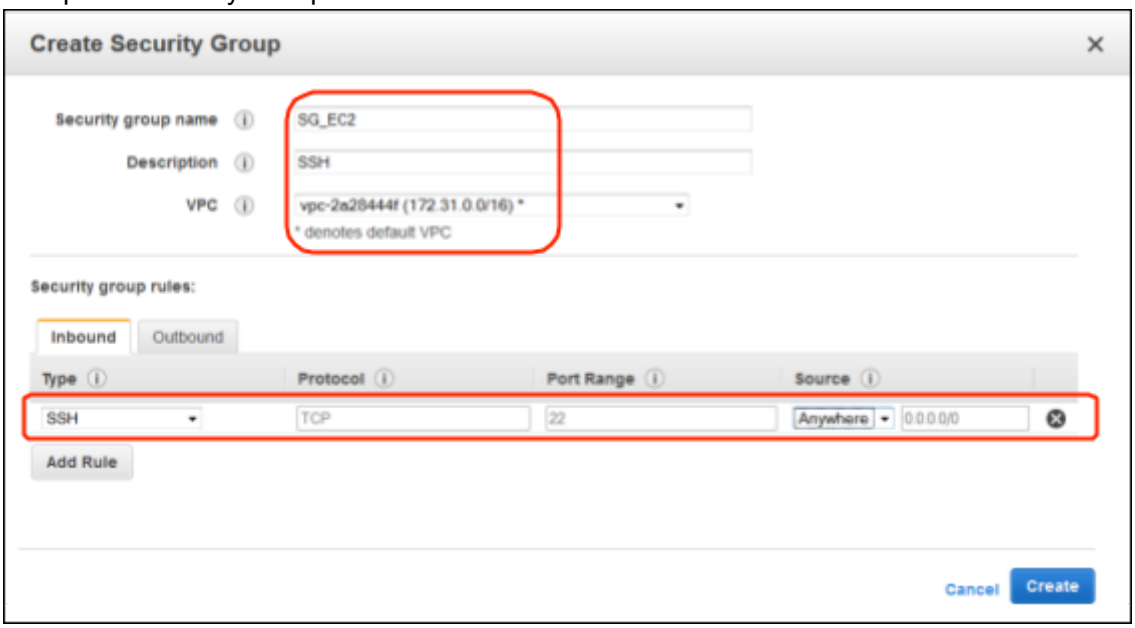
- Create a KeyPair called ‘key-ec2’ and **store it in a safe location**, you will need this key to connect to the instances.

If you are a Linux/MAC user, remember to set the priority for the key pair: chmod 700 <path-to-the-keypairs>.

3.5.3 Select “Security Groups” from the left menu, name a Security Group ‘SG_EC2’, add description and a SSH rule with ‘anywhere’ option selected in source field.

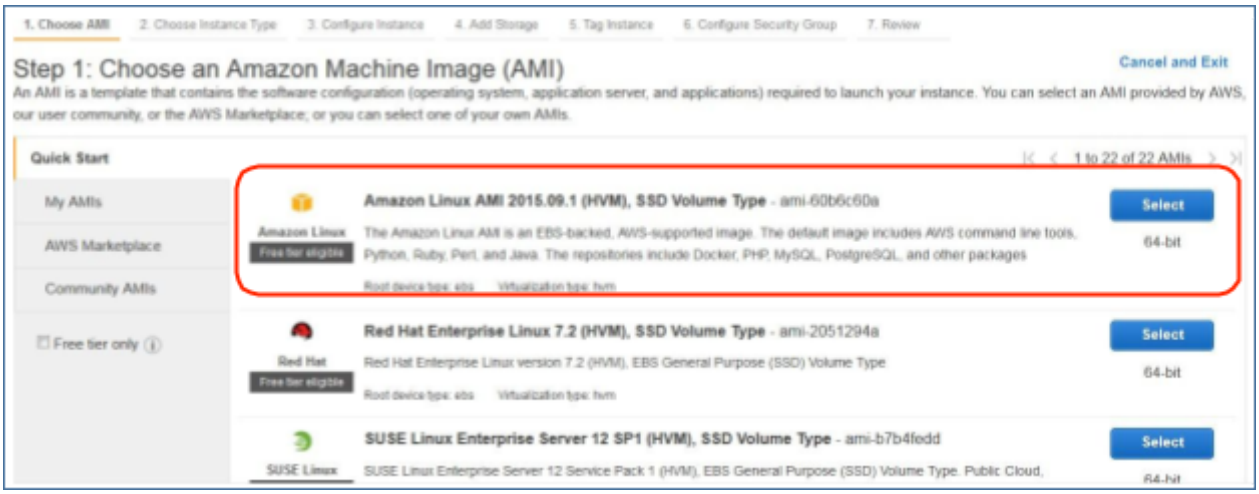


- Example of Security Group creation.

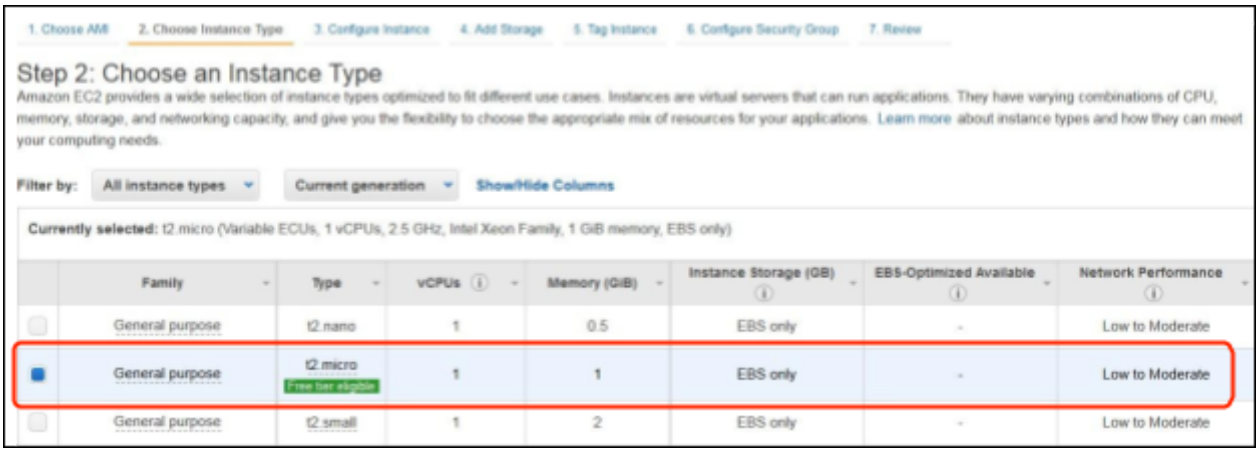


3.5.4 Launch your Instance

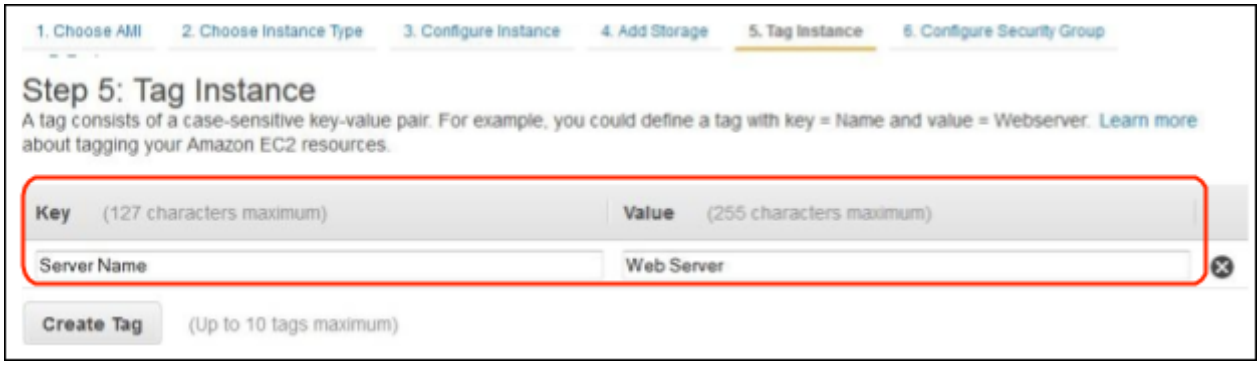
- In left menu, launch a new instance in the ‘Instances’ option Click on the ‘Launch Instance’ button and select the first Image of ‘Amazon Linux’ on the list



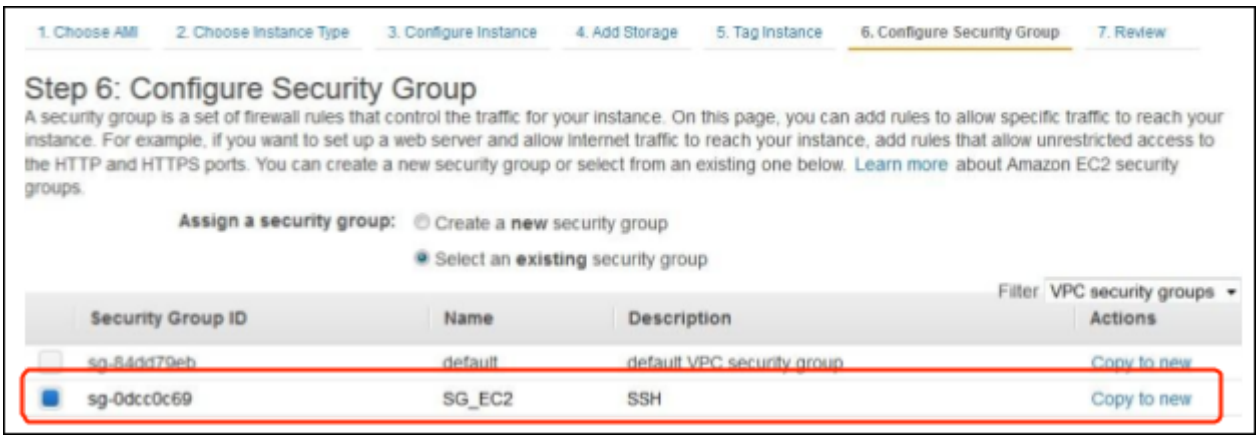
- Select the t2.micro instance that is ‘Free’.



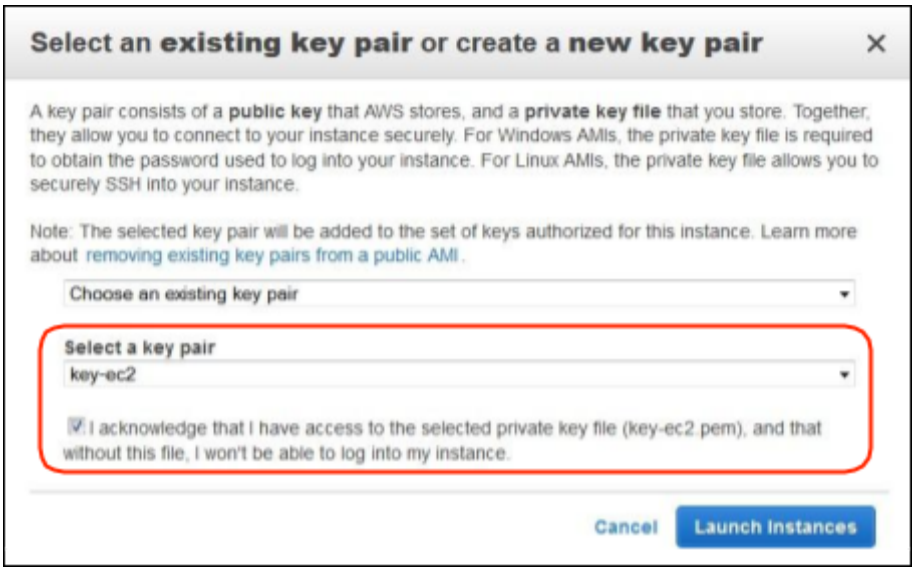
- Keep default values in the next configuration windows and continue until you get to the ‘Tag Instance’ option. Add ‘Key’ and ‘Value’ as shown in figure below and click on ‘Next: Configure Security Group’.



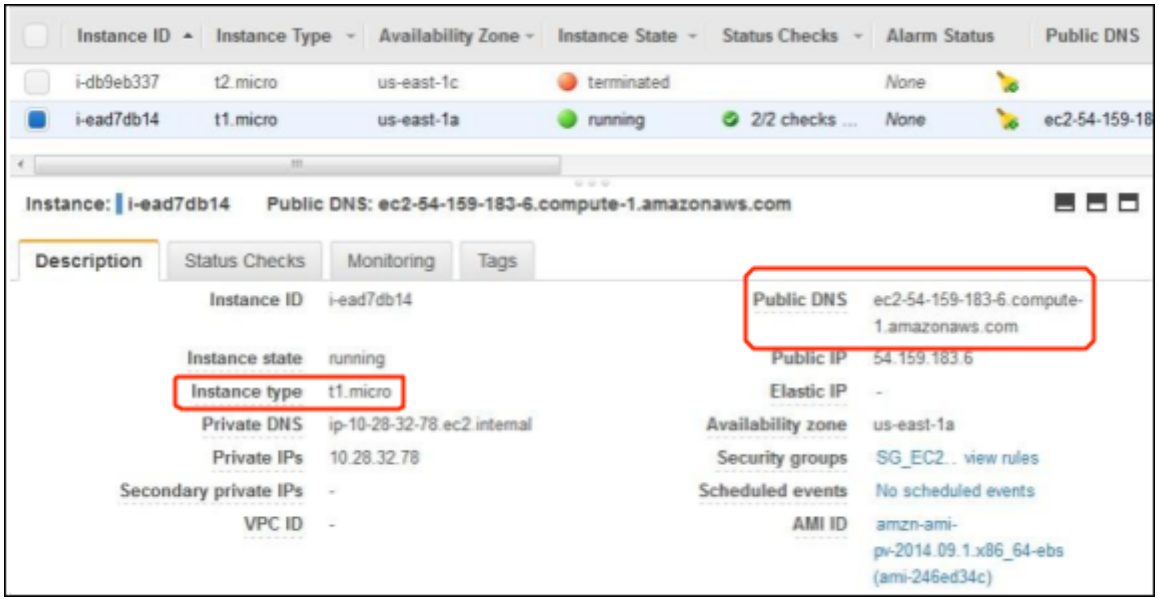
- Select the 'Security Group' created previously and click on 'Review and Launch'.



- Once you click 'launch' you will be prompted to choose the key pair 'key-ec2' created previously.

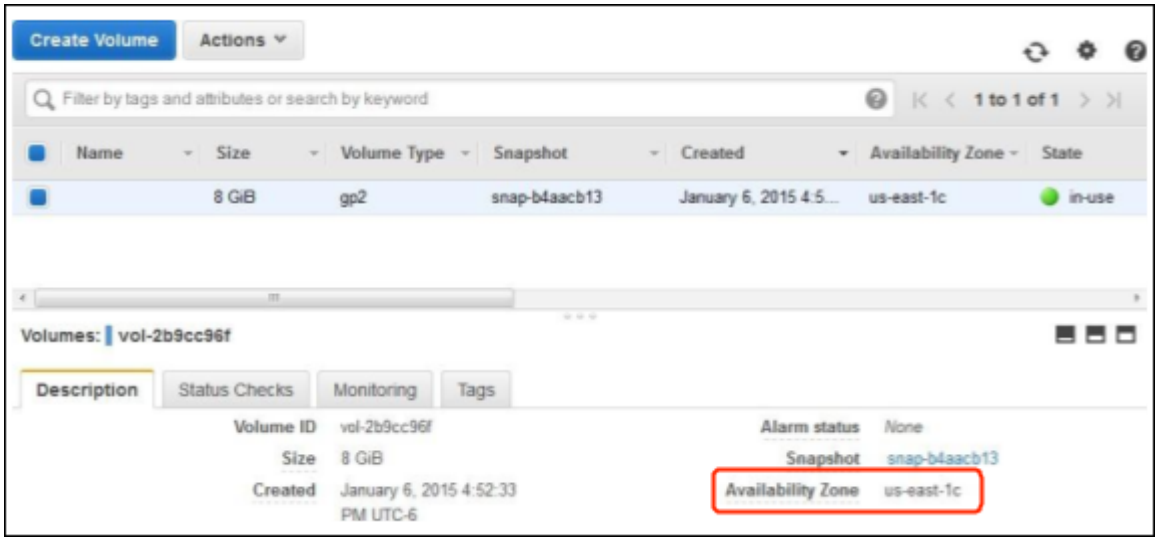


- In a short time your new instance will be deployed and ready to be used.

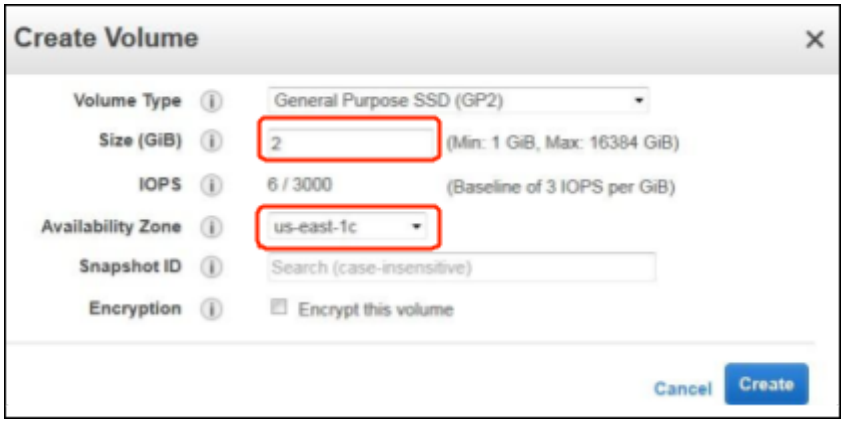


3.6 Adding volume

In left menu “Elastic Block Store”, select ‘Volumes’ and you will see the default volume where your instance is stored.



- Click in ‘Create Volume’ for creating a new **2GB** volume **and Volume Type ‘General Purpose (SSD)’**. Be sure that the ‘**Availability Zone**’ in the new volume is the same as the ‘Zone’ of the first volume (For this specific case us-east-1c)

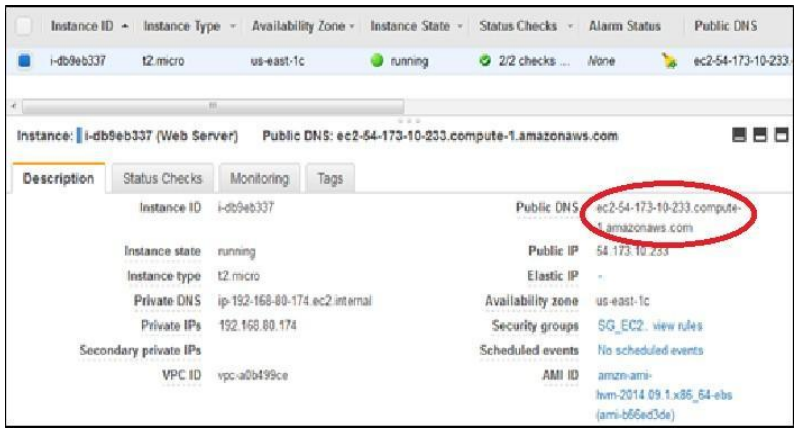


- Once the new volume is ‘available’, right click on it and select ‘Attach volume’, select your created instance and attach it. Note the path will be established in the ‘Device’ option.

- You will end up with a new 2 GB SDD storage drive attached to your instance.
- It is good practice to name your volumes.

3.7 Connection to the instance using SSH

First up, copy the public DNS.



3.7.1 For Linux and MAC OS

Open a terminal and type in:

```
ssh -i <path-to-your-key-pairs> <public DNS>
```

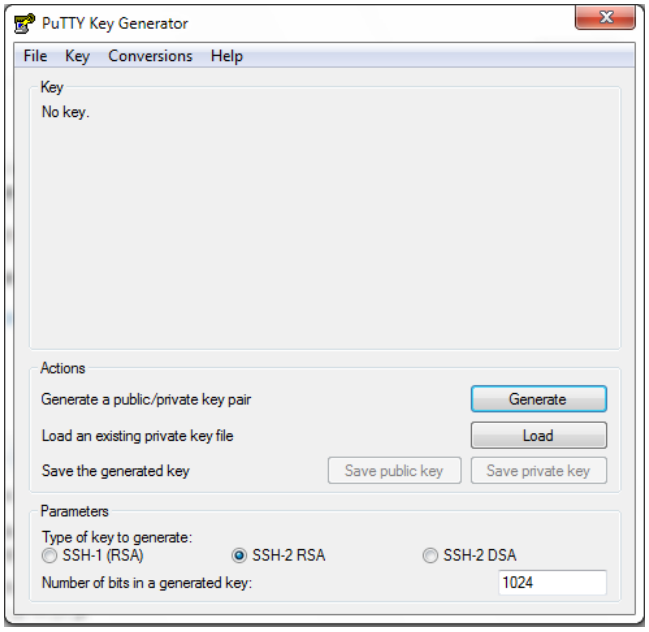
```
$ ssh -i key-ec2.pem ec2-user@ec2-54-173-10-233.compute-1.amazonaws.com
```

You will be logged in the amazon instance.

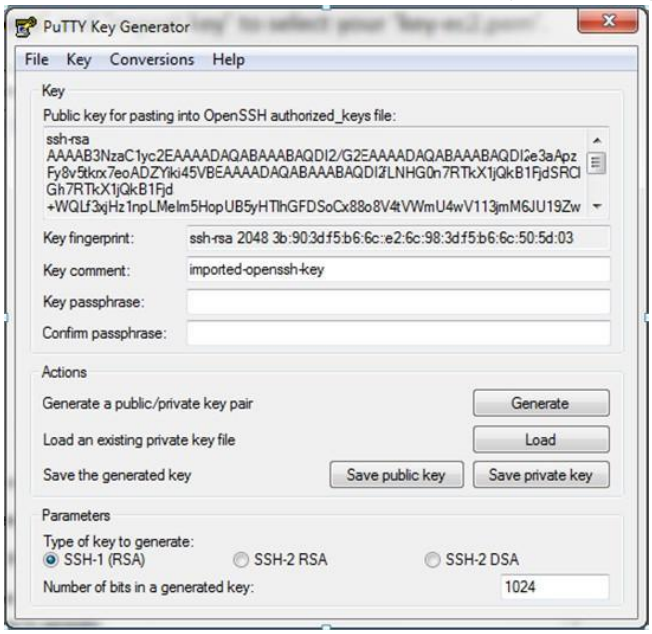
```
       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2014.09-release-notes/
18 package(s) needed for security, out of 42 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-192-168-80-174 ~]$
```

3.7.2 For Windows

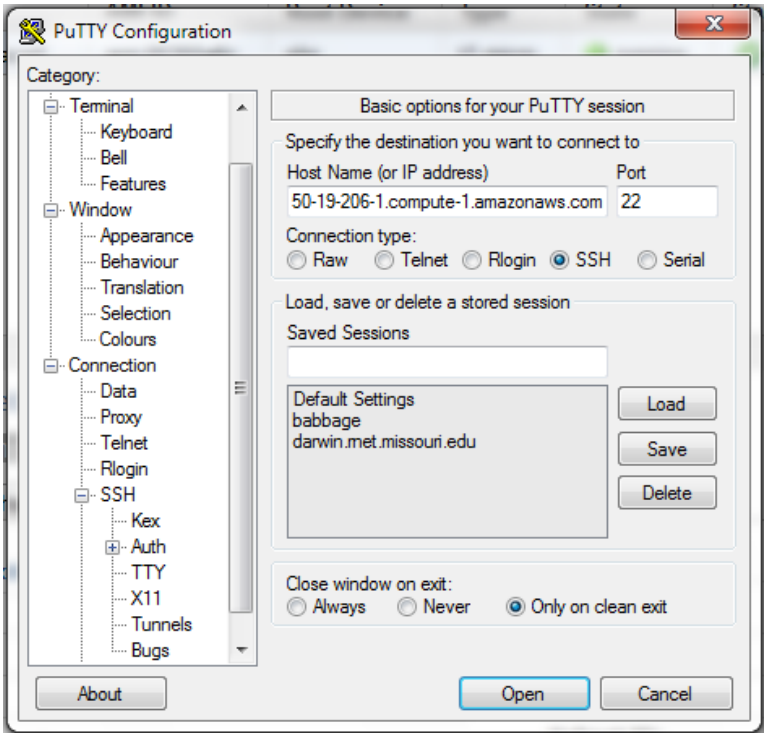
Download ‘PuTTY Key Generator’ to convert your key to PuTTY compatible format. Click in ‘conversions’ and ‘import key’ to select your ‘key-ec2.pem’.



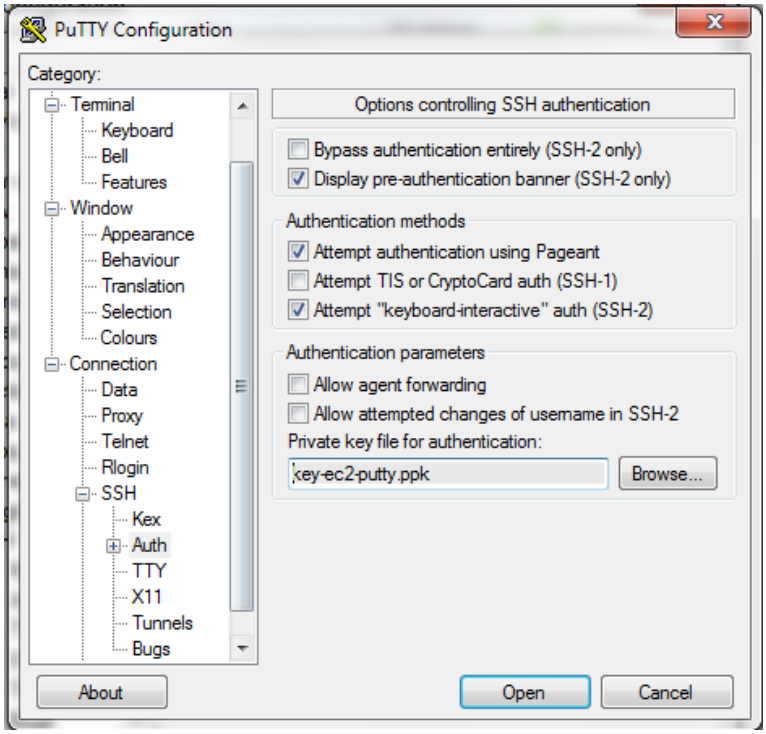
Select the ‘SSH-1 (RSA)’ check box and click on ‘Save private key’ for storing. Name it ‘key-ec2-putty’



Now with PuTTY. Paste [public dns] in ‘Host Name (or IP address)’



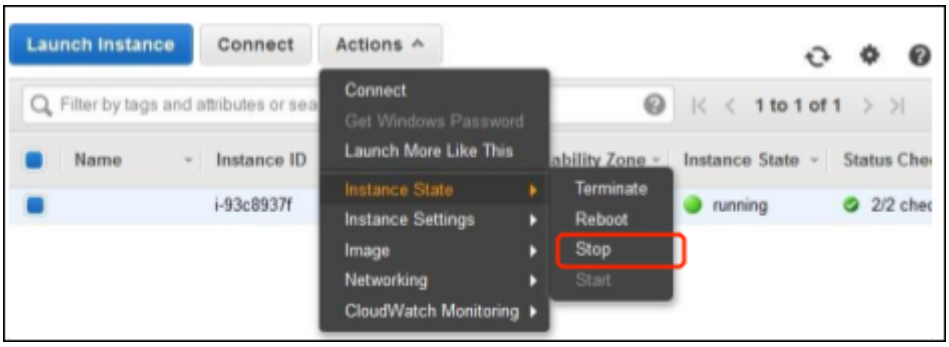
Browse the ‘key-ec2-putty’ in Connection/SSH/Auth then click Browse.



Select ‘Yes’ in the PuTTY alert. Then Login as ‘ec2-user’

3.8 You should stop your instance after finishing the lab.

In your AWS EC2 service select ‘Instances’ under ‘INSTANCES’ option, select your running instance, click on ‘Actions’ button and ‘Stop’ option.



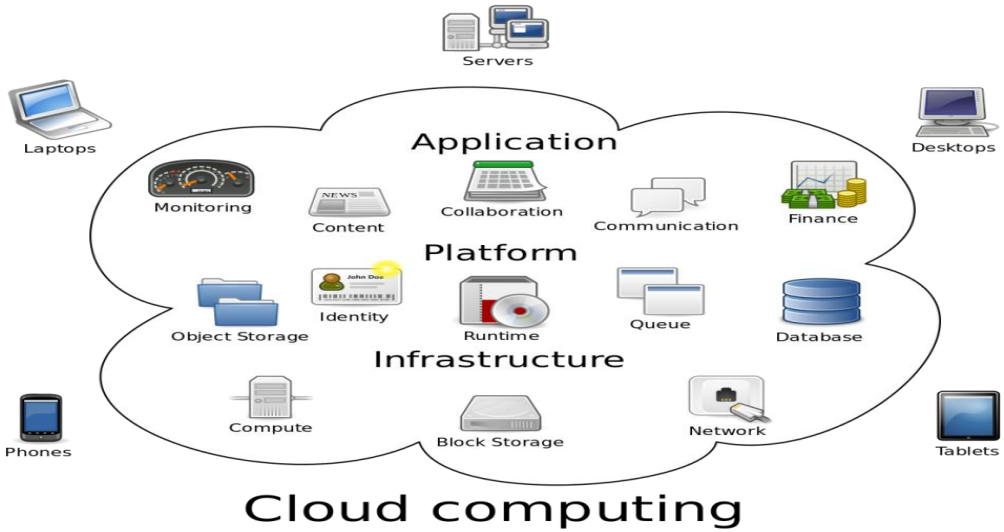
Introduction to Cloud Computing

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss a brief introduction to Cloud Computing and also focus on the aspects *i.e*. Why Clouds, What is a Cloud, Whats new in todays Clouds and also distinguish Cloud Computing from the previous generation of distributed systems.



Scalable Computing Over the Internet

- Evolutionary changes that have occurred in **distributed and cloud computing** over the past 30 years, **driven by applications with variable workloads and large data sets .**
- Evolutionary changes in machine architecture, operating system platform, network connectivity, and application workload.
- **Distributed computing** system uses multiple computers to solve large-scale problems over the Internet. Thus, **distributed computing becomes data-intensive and network-centric.**
- **The emergence of computing clouds** instead demands high-throughput computing (HTC) systems built with distributed computing technologies.
- **High-throughput computing (HTC)** appearing as computer clusters, service-oriented architecture, computational grids, peer-to-peer networks, Internet clouds, and the future Internet of Things.

The Hype of Cloud: Forecasting

- Gartner in 2009 – Cloud computing revenue will soar faster than expected and will **exceed $150 billion** by 2013. It will represent 19% of IT spending by 2015.
- IDC in 2009: "Spending on IT cloud services will triple in the next 5 years, reaching **$42 billion**."
- Forrester in 2010 – Cloud computing will go from **$40.7 billion** in 2010 to **$241 billion** in 2020.
- Companies and even federal/state governments using cloud computing now: **fbo.gov**

Many Cloud Providers

- AWS: Amazon Web Services
  - EC2: Elastic Compute Cloud
  - S3: Simple Storage Service
  - EBS: Elastic Block Storage
- Microsoft Azure
- Google Compute Engine/AppEngine
- Rightscale, Salesforce, EMC, Gigaspaces, 10gen, Datastax, Oracle, VMWare, Yahoo, Cloudera
- And 100s more…

Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- **Private clouds** are accessible only to company employees
- **Public clouds** provide service to any paying customer:
  - **Amazon S3 (Simple Storage Service):** store arbitrary datasets, pay per GB-month stored
  - **Amazon EC2 (Elastic Compute Cloud):** upload and run arbitrary OS images, pay per CPU hour used
  - **Google App Engine/Compute Engine:** develop applications within their App Engine framework, upload data that will be imported into their format, and run

Customers Save: Time and Money

- "With AWS, a new server can be up and running in **three minutes** compared to **seven and a half weeks** to deploy a server internally and a **64-node Linux cluster** can be online in five minutes (compared with three months internally."
- "With Online Services, reduce the IT **operational costs** by roughly **30%** of spending"
- "A private cloud of virtual servers inside its datacenter has saved nearly **crores of rupees annually**, because the company can share computing power and storage resources across servers."
- 100s of startups can harness large computing resources without buying their own machines.

What is a Cloud?

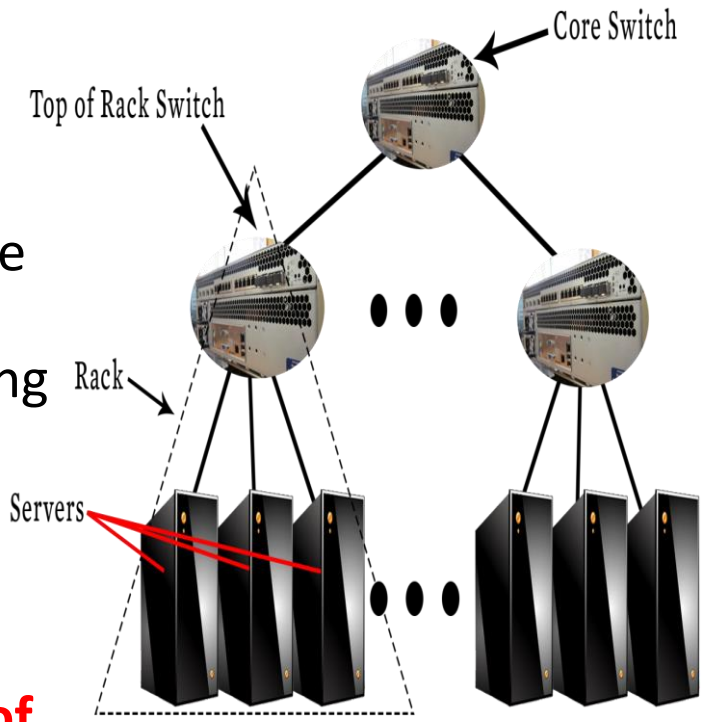
- Advances in virtualization make it possible to see the growth of Internet clouds **as a new computing paradigm**.
- i.e. dramatic differences between developing software for millions to use **as a service** versus distributing software to run on their PCs."

History:

- **In 1984, John Gage** Sun Microsystems gave the slogan, **"The network is the computer."**
- **In 2008, David Patterson** UC Berkeley said, **"The data center is the computer."**
- Recently, **Rajkumar Buyya** of Melbourne University simply said: **"The cloud is the computer."**
- Some people view **clouds as grids** or **clusters** with changes through virtualization, since clouds are anticipated to process huge data sets generated by the traditional Internet, social networks, and the future IoT.

What is a Cloud?

- **A single-site cloud (as known as "Datacenter") consists of**
  - Compute nodes (grouped into racks)
  - Switches, connecting the racks
  - A network topology, e.g., hierarchical
  - Storage (backend) nodes connected to the network
  - Front-end for submitting jobs and receiving client requests
  - (Often called "three-tier architecture")
  - Software Services

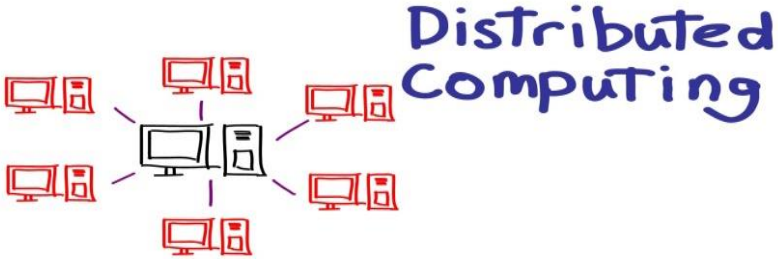


- **A geographically distributed cloud consists of**
  - Multiple such sites
  - Each site perhaps with a different structure and services

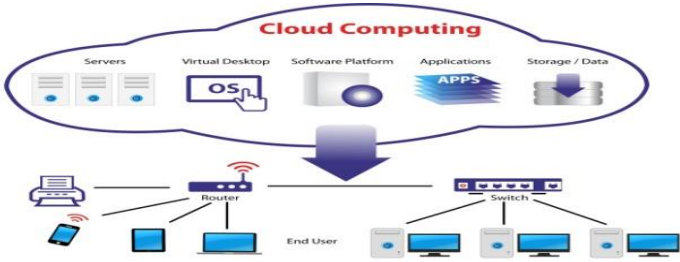
Computing Paradigm Distinctions

- Cloud computing overlaps with distributed computing.

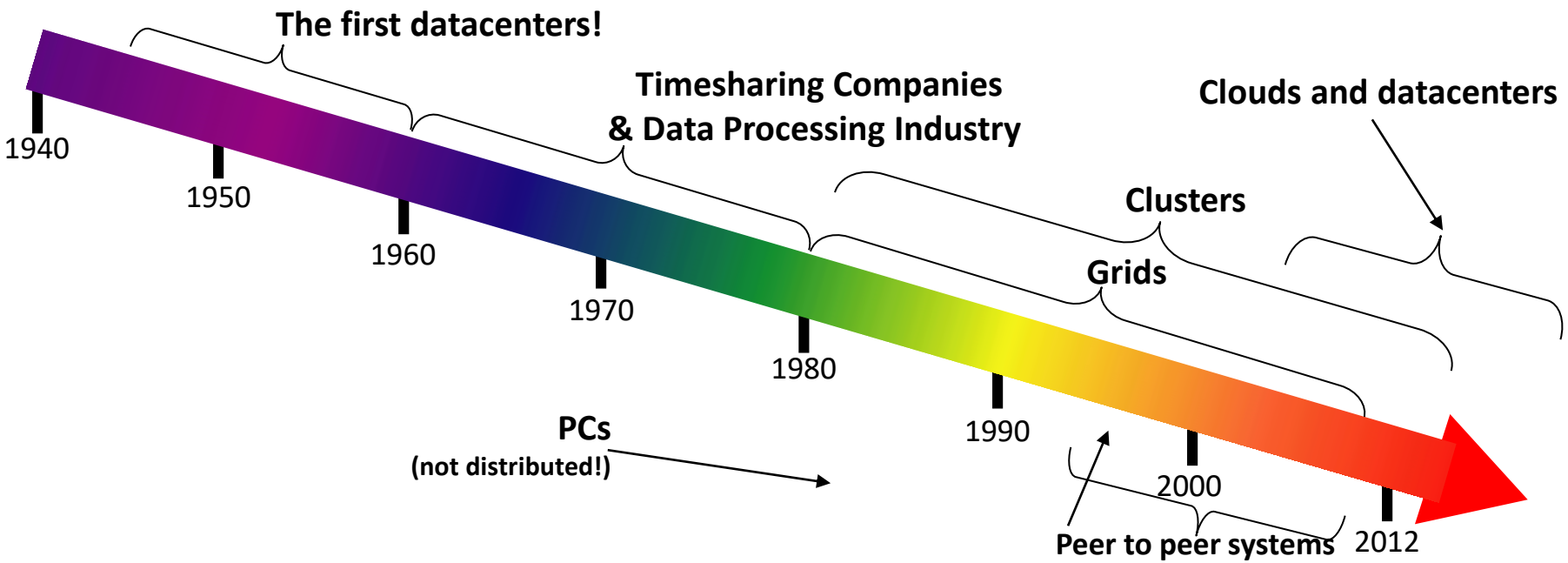
- **Distributed computing:** A ***distributed system*** consists of multiple autonomous computers, having its own memory, communicating through ***message passing***.



- **Cloud computing:** Clouds can be built with physical or virtualized resources over large data centers that are distributed systems. Cloud computing is also considered to be a form of ***utility computing or service computing.***



"A Cloudy History of Time"



"A Cloudy History of Time"

First large datacenters: ENIAC, ORDVAC, ILLIAC
Many used vacuum tubes and mechanical relays

1940

1950

1960

1970

1980

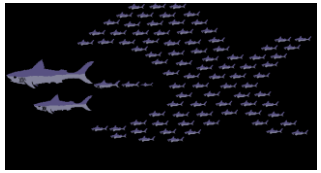
1990

2000

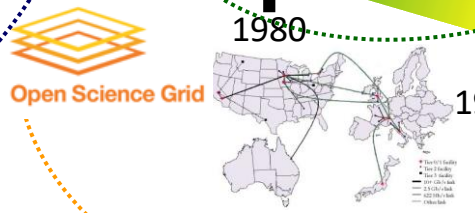
2012

Data Processing Industry
- 1968: $70 M. 1978: $3.15 Billion

Timesharing Industry (1975):
- Market Share: Honeywell 34%, IBM 15%,
- Xerox 10%, CDC 10%, DEC 10%, UNIVAC 10%
- Honeywell 6000 & 635, IBM 370/168, Xerox 940 & Sigma 9, DEC PDP-10, UNIVAC 1108



Berkeley NOW Project
Supercomputers
Server Farms (e.g., Oceano)



Grids (1980s-2000s):
- GriPhyN (1970s-80s)
- Open Science Grid and Lambda Rail (2000s)
- Globus & other standards (1990s-2000s)

P2P Systems (90s-00s)
- Many Millions of users
- Many GB per day

Clouds

Scalable Computing Trends: Technology

- **Doubling Periods** – storage: 12 months, bandwidth: 9 months, and CPU compute capacity: 18 months (what law is this?)
- **Moore's law** indicates that processor speed doubles every 18 months.
- **Gilder's law** indicates that network bandwidth has doubled each year in the past.

- Then and Now
  - Bandwidth
    - 1985: mostly 56Kbps links nationwide
    - 2015: Tbps links widespread
  - Disk capacity
    - Today's PCs have TBs, far more than a 1990 supercomputer

The Trend toward Utility Computing

- Aiming towards autonomic operations that can be self-organized to support dynamic discovery. Major computing paradigms are composable with ***QoS and SLAs (service-level agreements).***
- In 1965, MIT's Fernando Corbató of the Multics operating system envisioned a computer facility operating “like a power company or water company”.
- **Plug** your thin client into the computing Utility **and Play** Intensive Compute & Communicate Application
- **Utility computing** focuses on a business model in which customers receive computing resources from a paid service provider.
- All **grid/cloud platforms are regarded as utility service providers**.

Features of Today's Clouds

I. **Massive scale:** Very large data centers, contain tens of thousands sometimes hundreds of thousands of servers and you can run your computation across as many servers as you want and as many servers as your application will scale.

II. **On-demand access:** Pay-as-you-go, no upfront commitment.

- And anyone can access it

III. **Data-intensive Nature:** What was MBs has now become TBs, PBs and XBs.

- Daily logs, forensics, Web data, etc.

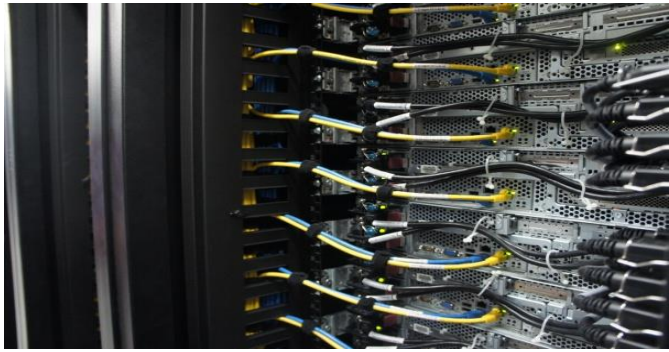
IV. **New Cloud Programming Paradigms:** MapReduce/Hadoop, NoSQL/Cassandra/MongoDB and many others.

- **Combination of one or more of these gives rise to novel and unsolved distributed computing problems in cloud computing.**

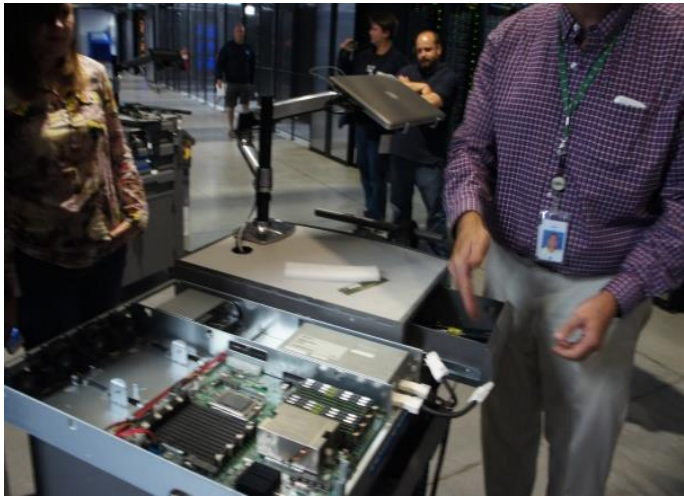
I. Massive Scale

- **Facebook [GigaOm, 2012]**
  - 30K in 2009 -> 60K in 2010 -> 180K in 2012
- **Microsoft [NYTimes, 2008]**
  - 150K machines
  - Growth rate of 10K per month
  - 80K total running Bing
  - In 2013, Microsoft Cosmos had 110K machines (4 sites)
- **Yahoo! [2009]:**
  - 100K
  - Split into clusters of 4000
- **AWS EC2 [Randy Bias, 2009]**
  - 40K machines
  - 8 cores/machine
- **eBay [2012]: 50K machines**
- **HP [2012]: 380K in 180 DCs**
- **Google: A lot**

What does a datacenter look like from inside?



Lots of Servers



Power and Energy

- WUE = Annual Water Usage / IT Equipment Energy (L/kWh)
  - low is good
- PUE = Total facility Power / IT Equipment Power
  - low is good (e.g., Google~1.11)

Off-site

On-site

Cooling

- Air sucked in
- Combined with purified water
- Moves cool air through system

II. On-demand access: *AAS Classification

- **On-demand:** renting vs. buying one. E.g.:
  - AWS Elastic Compute Cloud (EC2): a few cents to a few $ per CPU hour
  - AWS Simple Storage Service (S3): a few cents per GB-month
- **HaaS: Hardware as a Service**
  - Get access to barebones hardware machines, do whatever you want with them, Ex: Your own cluster
  - Not always a good idea because of security risks
- **IaaS: Infrastructure as a Service**
  - Get access to flexible computing and storage infrastructure. **Virtualization** is one way of achieving this. subsume HaaS.
  - Ex: Amazon Web Services (AWS: EC2 and S3), OpenStack, Eucalyptus, Rightscale, Microsoft Azure, Google Cloud.

II. On-demand access: *AAS Classification

- **PaaS: Platform as a Service**
  - Get access to flexible computing and storage infrastructure, coupled with a software platform (often tightly coupled)
  - Ex: Google's AppEngine (Python, Java, Go)

- **SaaS: Software as a Service**
  - Get access to software services, when you need them. subsume SOA (Service Oriented Architectures).
  - Ex: Google docs, MS Office on demand

III. Data-intensive Computing

- **Computation-Intensive Computing**
  - Example areas: MPI-based, High-performance computing, Grids
  - Typically run on supercomputers (e.g., NCSA Blue Waters)
- **Data-Intensive**
  - Typically store data at datacenters
  - Use compute nodes nearby
  - Compute nodes run computation services

- In data-intensive computing, the **focus shifts from computation to the data**:
- CPU utilization no longer the most important resource metric, instead I/O is (disk and/or network)

IV. New Cloud Programming Paradigms

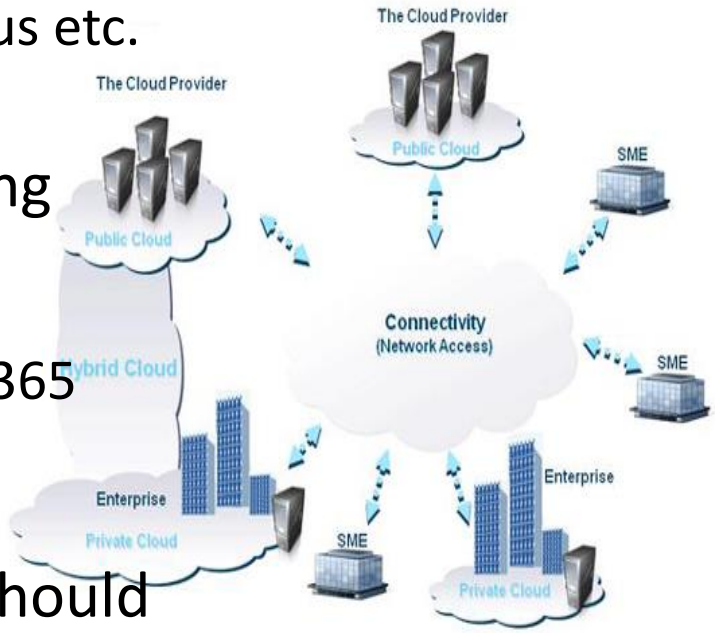
- Easy to write and run highly parallel programs in new cloud programming paradigms:
  - **Google:** MapReduce and Sawzall
  - **Amazon:** Elastic MapReduce service (pay-as-you-go)
  - Google (MapReduce)
    - Indexing: a chain of 24 MapReduce jobs
    - ~200K jobs processing 50PB/month (in 2006)
  - **Yahoo!** (Hadoop + Pig)
    - WebMap: a chain of several MapReduce jobs
    - 300 TB of data, 10K cores, many tens of hours (~2008)
  - **Facebook** (Hadoop + Hive)
    - ~300TB total, adding 2TB/day (in 2008)
    - 3K jobs processing 55TB/day
  - NoSQL: MySQL is an industry standard, but Cassandra is 2400 times faster

Two Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud

- **Private clouds** are accessible only to company employees
- Example of popular vendors for creating private clouds are VMware, Microsoft Azure, Eucalyptus etc.

- **Public clouds** provide service to any paying customer
- Examples of large public cloud services include Amazon EC2, Google AppEngine, Gmail, Office365 and Dropbox etc.



- You're starting a new service/company: should you use a public cloud or purchase your own private cloud?

Single site Cloud: to Outsource or Own?

- Medium-sized organization: wishes to run a service for *M* months
  - Service requires 128 servers (1024 cores) and 524 TB
- **Outsource** (e.g., via AWS): *monthly* cost
  - S3 costs: $0.12 per GB month. EC2 costs: $0.10 per CPU hour (costs from 2009) Storage = $ 0.12 X 524 X 1000 ~ $62 K
  - Total = Storage + CPUs = $62 K + $0.10 X 1024 X 24 X 30 ~ $136 K
- **Own**: monthly cost
  - Storage ~ $349 K / *M* Total ~ $ 1555 K / *M* + 7.5 K (includes 1 sysadmin / 100 nodes)
    - using 0.45:0.4:0.15 split for hardware:power: network and 3 year lifetime of hardware
- Breakeven analysis: **more preferable to own if:**
  - $349 K / *M* < $62 K (storage)
  - $ 1555 K / *M* + 7.5 K < $136 K (overall)

  Breakeven points
  - *M* > 5.55 months (storage)
  - *M* > 12 months (overall)

-Startups use clouds a lot
-Cloud providers benefit monetarily most from storage

Conclusion

- Clouds build on many previous generations of distributed systems

- Characteristics of cloud computing problem

  - **Scale, On-demand access, data-intensive, new programming**

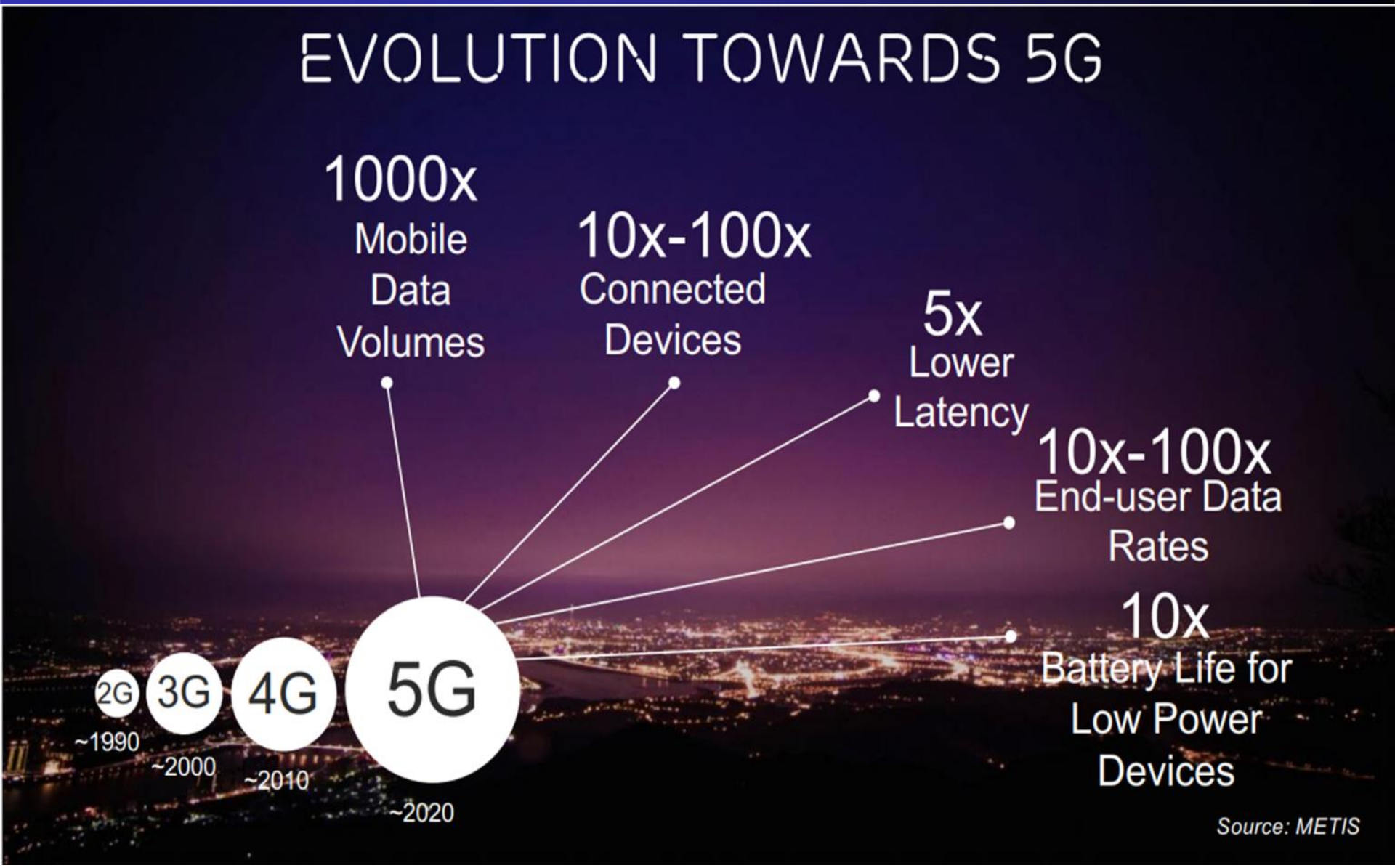
Cloud Native 5G for Industrial Applications

Dr. Rajiv Misra
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

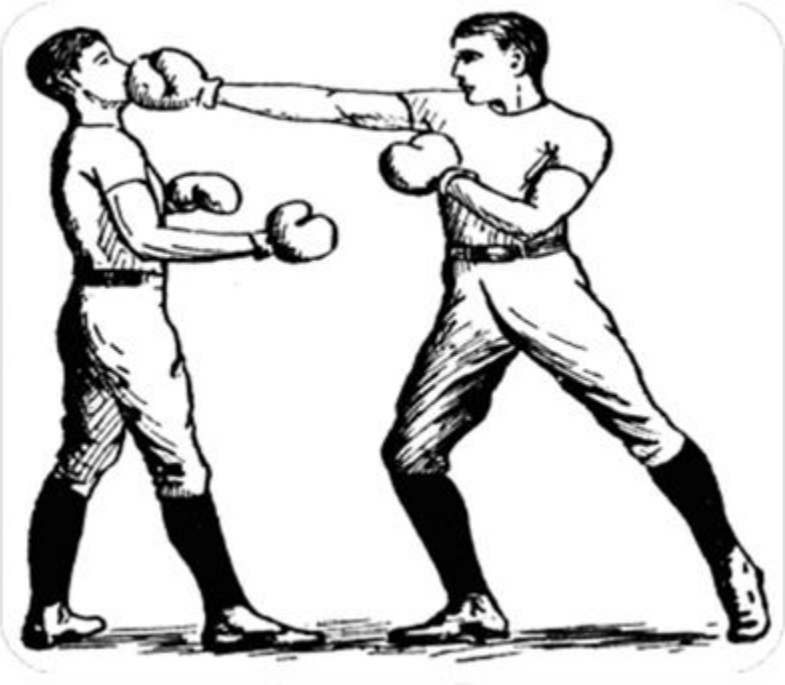
Contents

1. Cloud Computing to Cloud (Native) Computing
2. 5G to Cloud (Native) 5G
3. Cloud Native 5G (non-4G) Application
4. Conclude

5G


EVOLUTION TOWARDS 5G
1000x
Mobile
Data
Volumes
10x-100x
Connected
Devices
5x
Lower
Latency
10x-100x
End-user Data
Rates
10x
Battery Life for
Low Power
Devices
2G
3G
4G
5G
~1990
~2000
~2010
~2020
Source: METIS

The Frontier of Networking



Existing

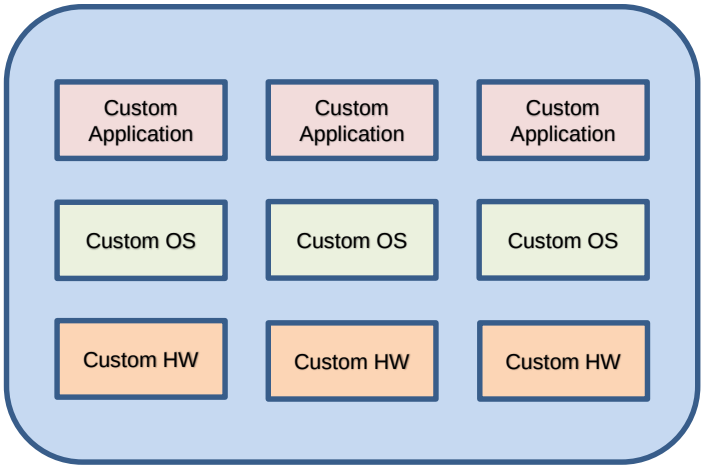
- CLIs
- Closed Source
- Vendor Lead
- Classic Network Appliances

New

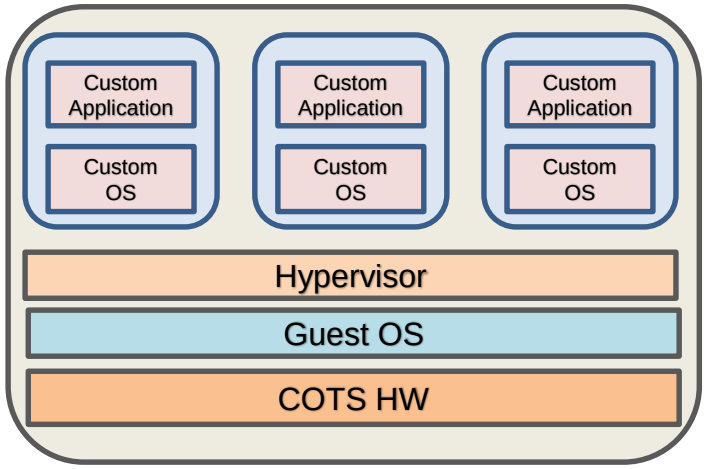
- APIs
- Open Source
- Customer Lead
- Network Function Virtualization (NFV)

Cloud Computing: Virtualization

- Traditionally Mobile core application software runs on proprietary hardware. This HW is deployed OEMs/Vendors specifically for custom applications.
- Such HW is optimized for speed and performance and has a fixed capacity. Capacity increases often require HW swaps or adding additional HW resources (CPU, memory, storage).
- The current model does not scale well and is not cost effective.
- Virtualization enables running applications on virtual machines, which run on COTS HW. This enables the decoupling of application and HW and therefore offers great flexibility at a significantly reduced cost.



Traditional Architecture



VM Based Architecture

Cloud Native

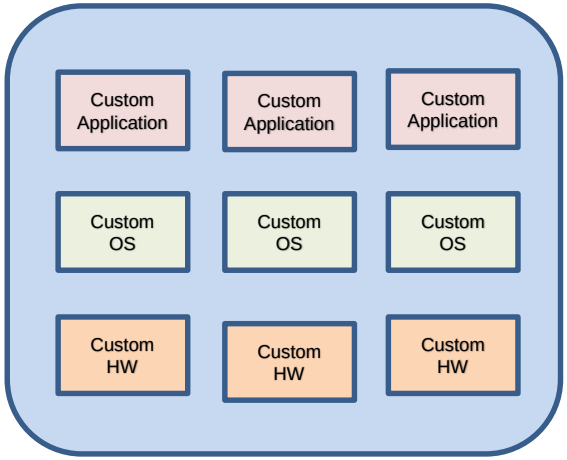
- Cloud Native architectures have gained a lot of attention lately as operators try to follow the same architecture principles as followed by hyperscalers like Amazon, Google etc.

- Cloud Native is not a single concept but instead is a set of principles. Some of these are:

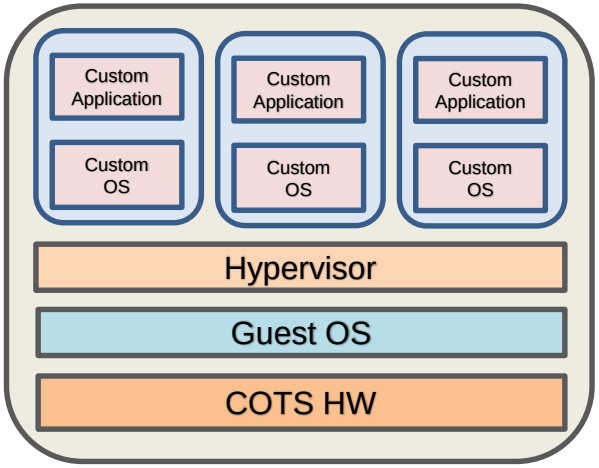
  - Infrastructure agnostic - Apps don't depend on HW and resources.
  - Software decomposition and Life Cycle Management - Apps run in smaller and manageable pieces unlike a monolithic application.
  - Resiliency - due to distributed nature of applications impact of local maintenance or faults is isolated to local instances and does not affect overall functionality.
  - Orchestration and Automation - Apps can be managed using orchestrators like Kubernetes or OpenStack.
  - Turn-up/down, scaling and maintenance (upgrades, logging etc.) of apps is all automated.

Containers

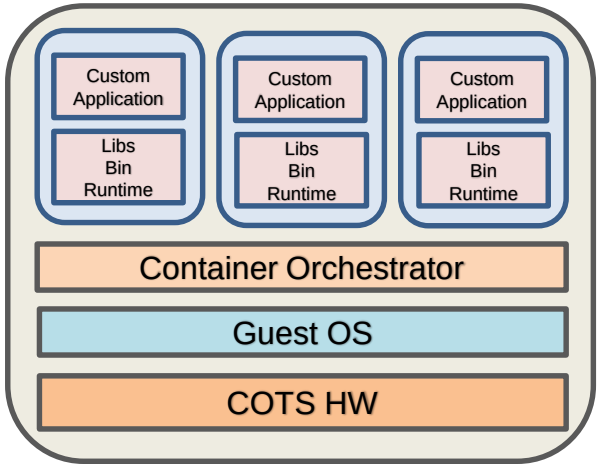
- Containers are independent hosts for applications that use a single, stripped-down version of an operating system to run.
- Virtual machines use a full version of an operating system.
- Containers run a virtualized workload, processed by an application broken up into microservices, making them more lightweight and flexible than a VM.
- VMs can run a full, unaltered application, orchestrated by a hypervisor.
- Both can scale up and down quickly and easily.



Traditional Architecture



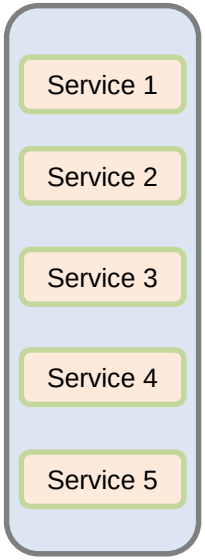
VM Based Architecture



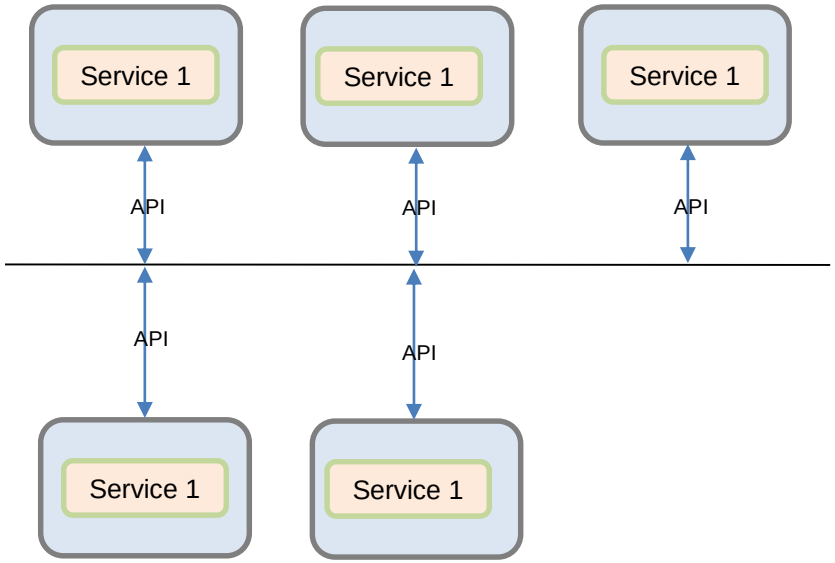
Container Based Architecture

Microservices

- Refers to an architectural and organizational approach to software development where application is composed of smaller indepedent services that interact with each other over well defined APIs.
- Form the basis for service based architecture (SBA).
- Several benefits:
  - Components have a limited scope and therefore changes can be made quickly and efficiently.
  - Instances can be added, removed on demand adding ease of scalability.
  - Independent software upgrades.
  - Ease of debugging due to limited scope of components.



Monolithic Architecture



Microservices based Architecture

Automation

- 5G needs to support rapid scaling up/down and life cycle management of network applications (turn up/down, upgrade, logging etc.)
- Automation technologies for orchestration can help achieve these goals.
- So far SON (self-optimizing networks) capabilities have been leverage on the RAN side. For example - Automatic Neighbor relations.
- Operators have seen accelerated roll out times, simplified network upgrades, fewer dropped calls, improved call setup success rates among other positive impacts. Therefore there is a great deal of interest to implement SON/automation on the core side as well.
- 5G is a unique architecture that is service-based and built on top of microservices. This offers a unique opportunity to leverage automation to manage life cycle of applications and infrastructure resources. Such capabilities are essential for supporting 5G use cases that demand varying demands/requirements.
- Machine learning and artificial intelligence will become further integrated across all aspects of mobile systems in the near future.

New Verticals

5G is 3GPP's answer for enabling new use cases

- beyond smartphones and traditional IoT applications
- industrial AR, remote control of drones/farming vehicles, support for new vehicular services (NR-V2X), etc.

Address requirements not satisfied by today's WiFi or 4G-based solutions

- network density/scalability, resilience for critical communications

Private 5G

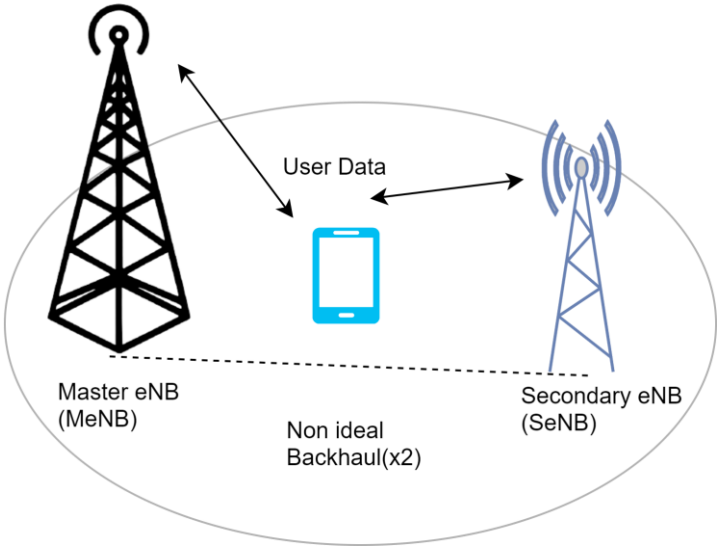
- evolution of private LTE exploiting the new 5G features (i.e. low-latency and ultra-reliable transmission, service classification through slicing)
- licensed bands for non-public applications (e.g. Industry 4.0)
- Customized IT + Radio solution (OS+Servers+Radio+Edge Computing)

5G, Small-cells, Dual Connectivity

Broadband low-latency → **more bandwdith** → **higher frequency** → **shorter range**

↓

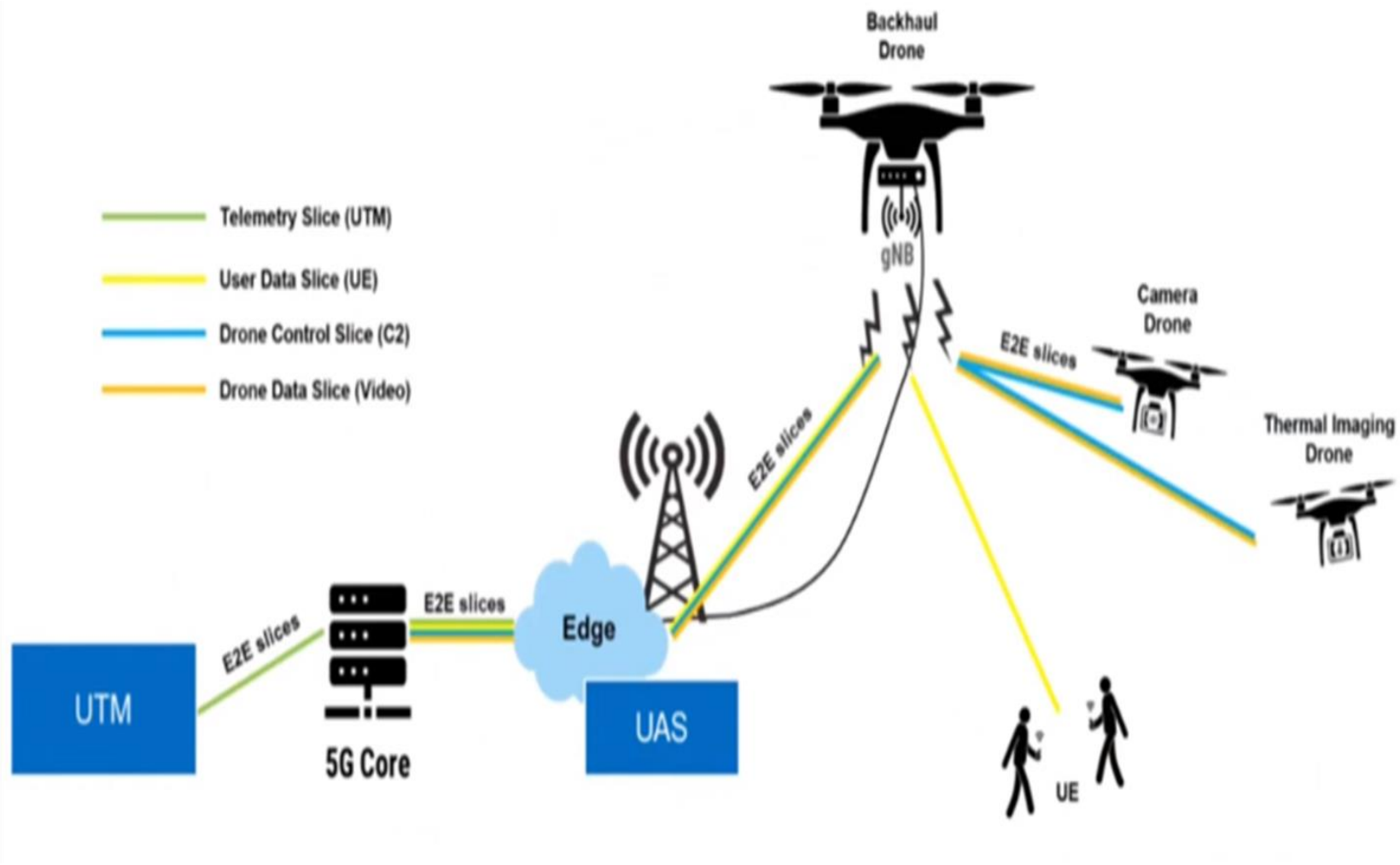
dual connectivity



user-plane from both
control-plane only from Master

→ almost always in coverage (4G), high-speed (5G) when near the SgNB)

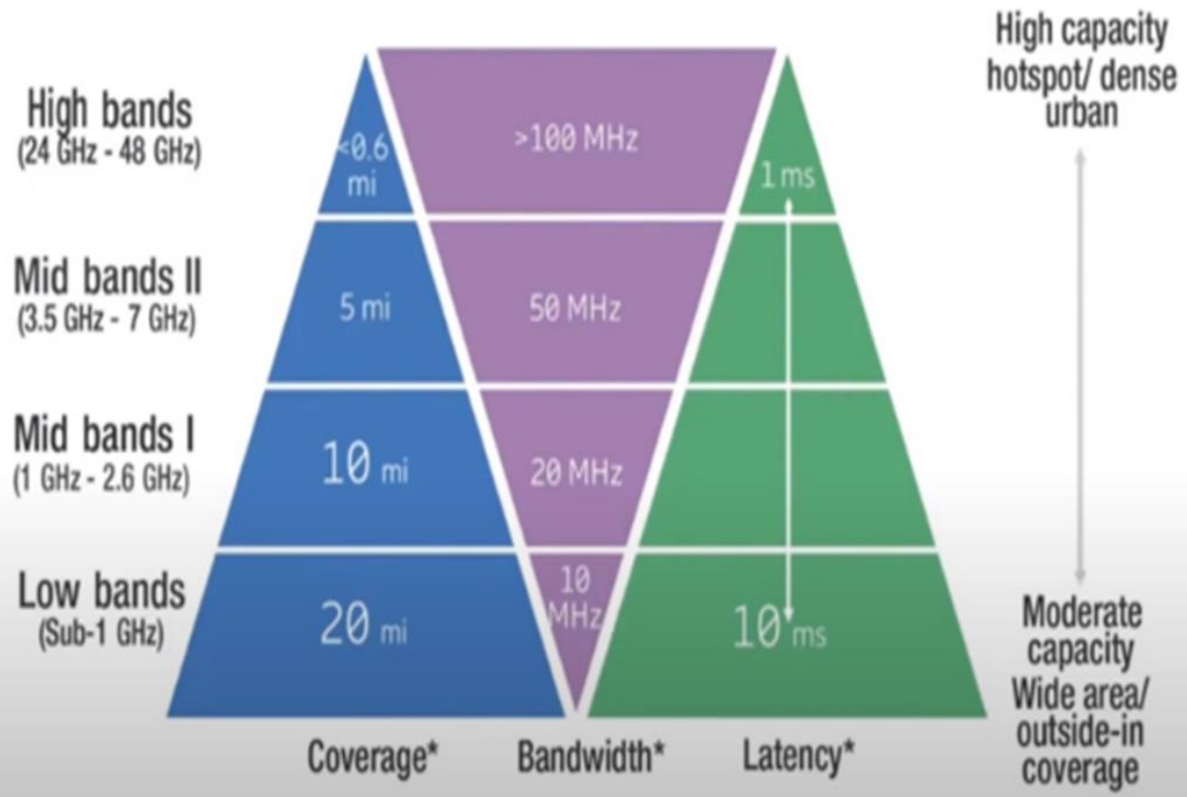
Public Safety: Forest Fire



Types of 5G

Types of 5G

- Not all 5G is created equal
- Choose 2: Coverage, bandwidth, Latency

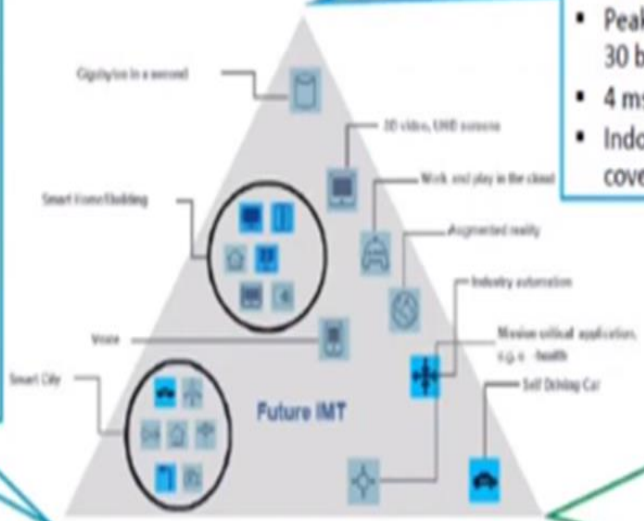


5G Classifications

Key challenge for 5G design: support for different services having diverging requirements

Massive Machine Type Communications (mMTC)

- Low data rates (1 to 100 kbps)
- High device density (up to 1,000,000 /km^2)
- Latency: Seconds to hours
- Low power: Up to 15 years battery life



Source: ITU-R SG5 WP-5D

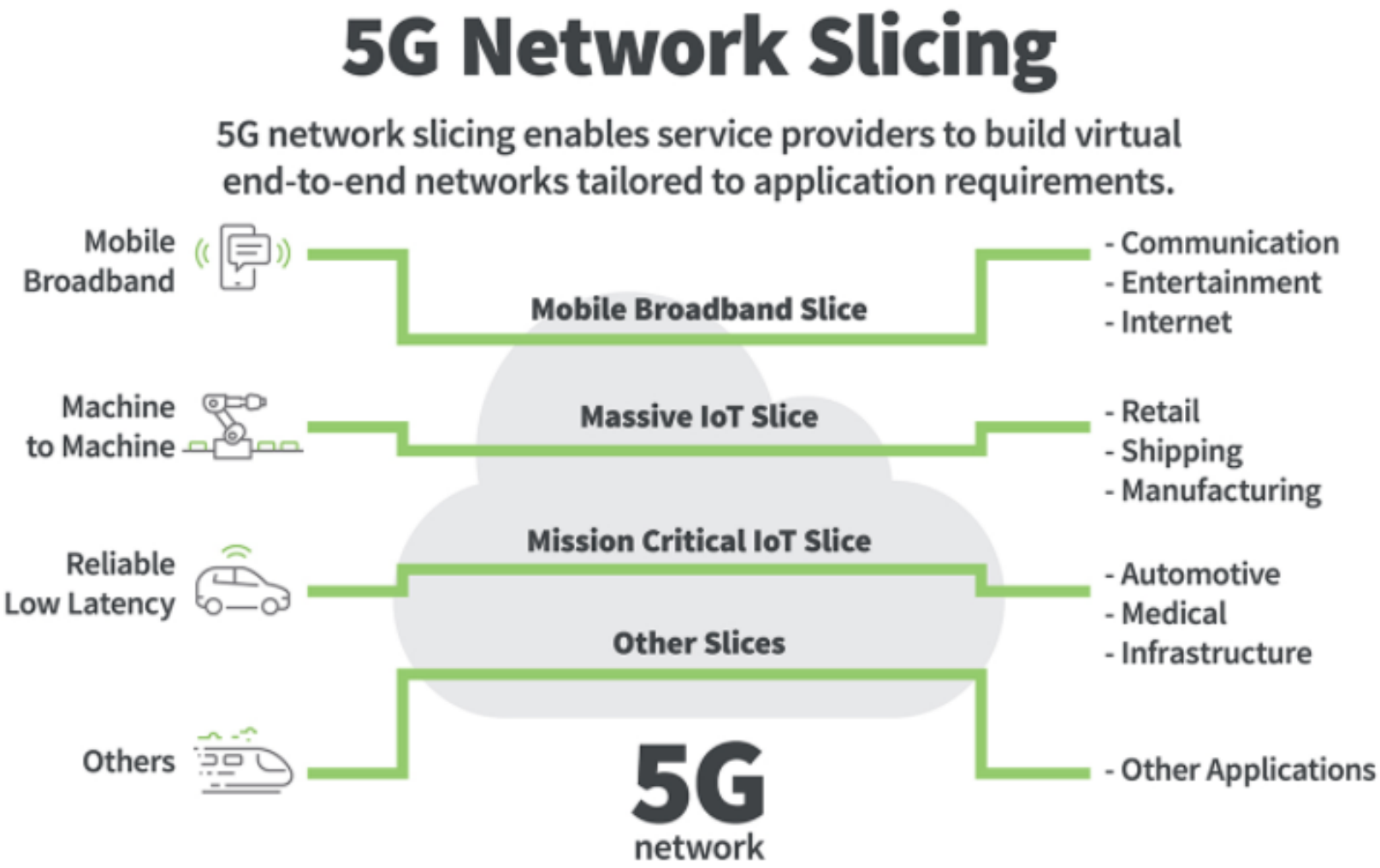
Enhanced Mobile Broadband (eMBB)

- Peak data rates: 20 Gbps (DL) and 10 Gbps (UL)
- Peak spectral efficiency: 30 bps/Hz (DL) and 15 bps/Hz (UL)
- 4 ms user plane latency
- Indoor/hotspot and enhanced wide-area coverage

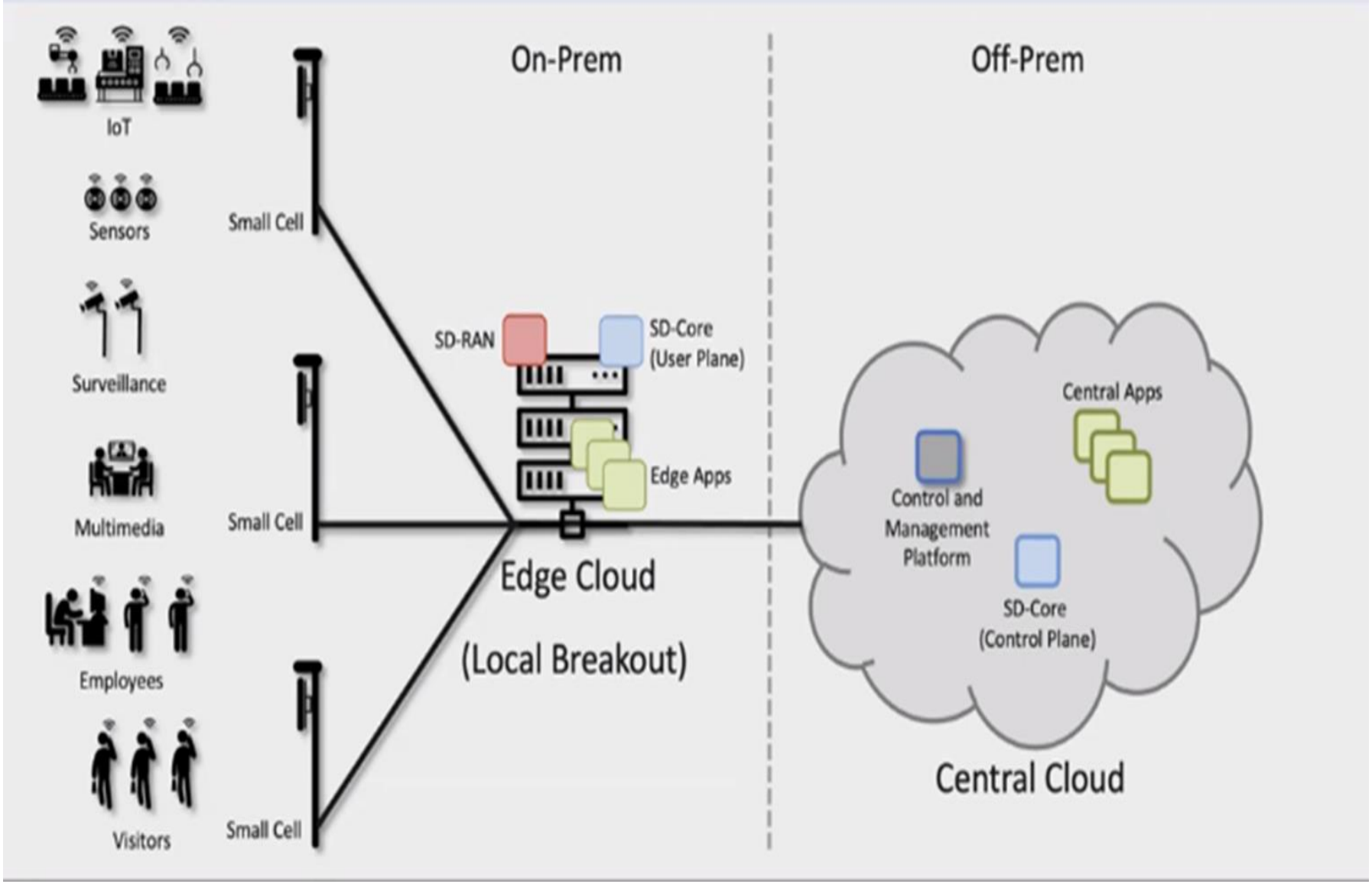
Ultra-Reliable and Low Latency Communications (URLLC)

- Low to medium data rates (50 kbps to 10 Mbps)
- 0.5 ms user plane latency
- 99.999% reliability and availability within 1 ms
- High mobility

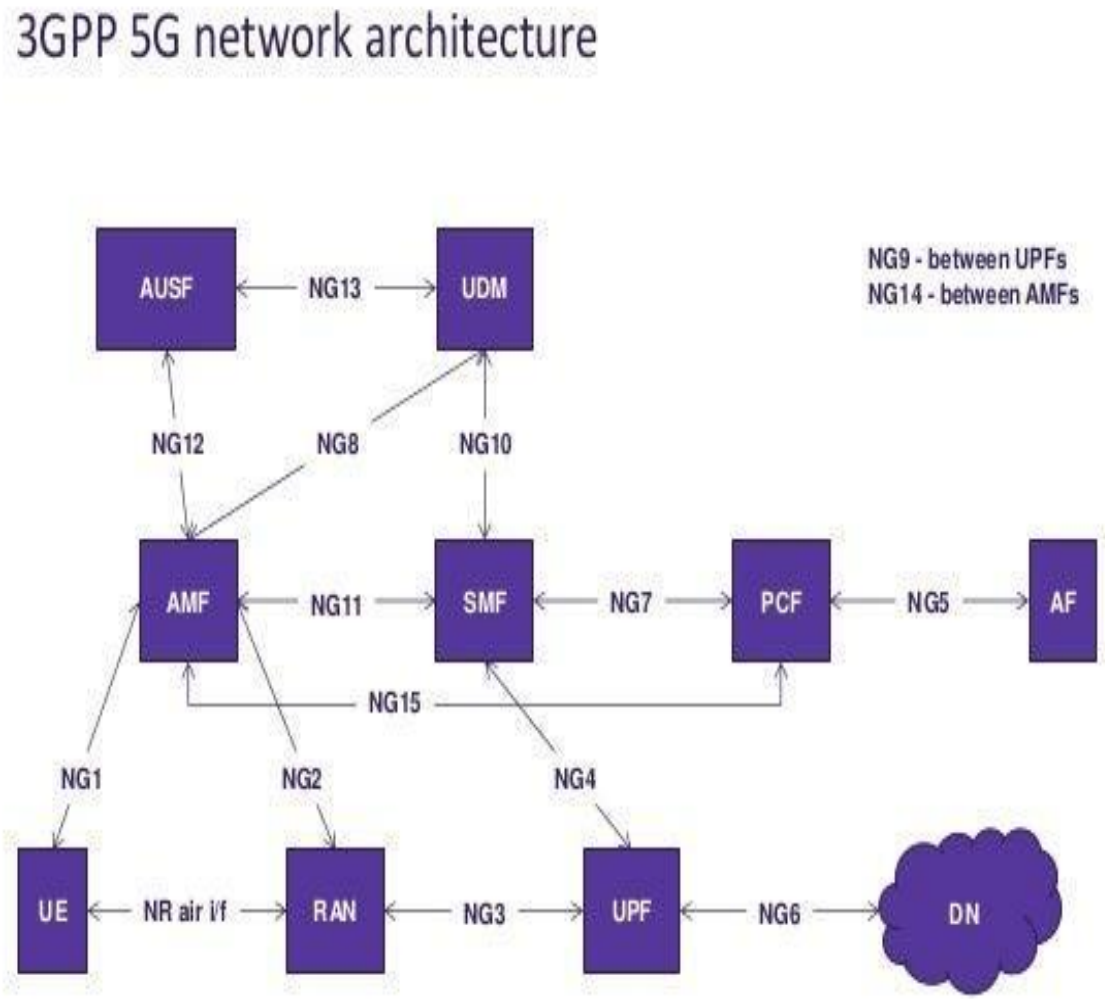
5G Network Slicing



Overview


On-Prem
Off-Prem
IoT
Sensors
Surveillance
Multimedia
Employees
Visitors
Small Cell
Small Cell
Small Cell
SD-RAN
SD-Core
(User Plane)
Edge Apps
Edge Cloud
(Local Breakout)
Central Apps
Control and
Management
Platform
SD-Core
(Control Plane)
Central Cloud

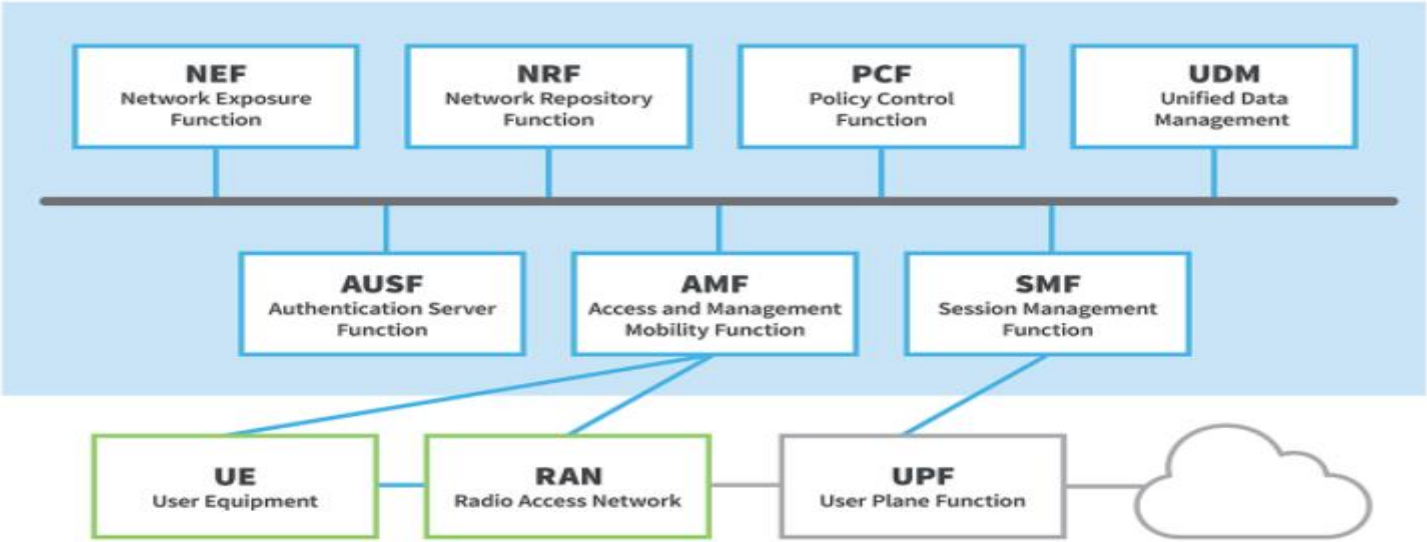
5G Network architecture



The components of the 5G core architecture include:

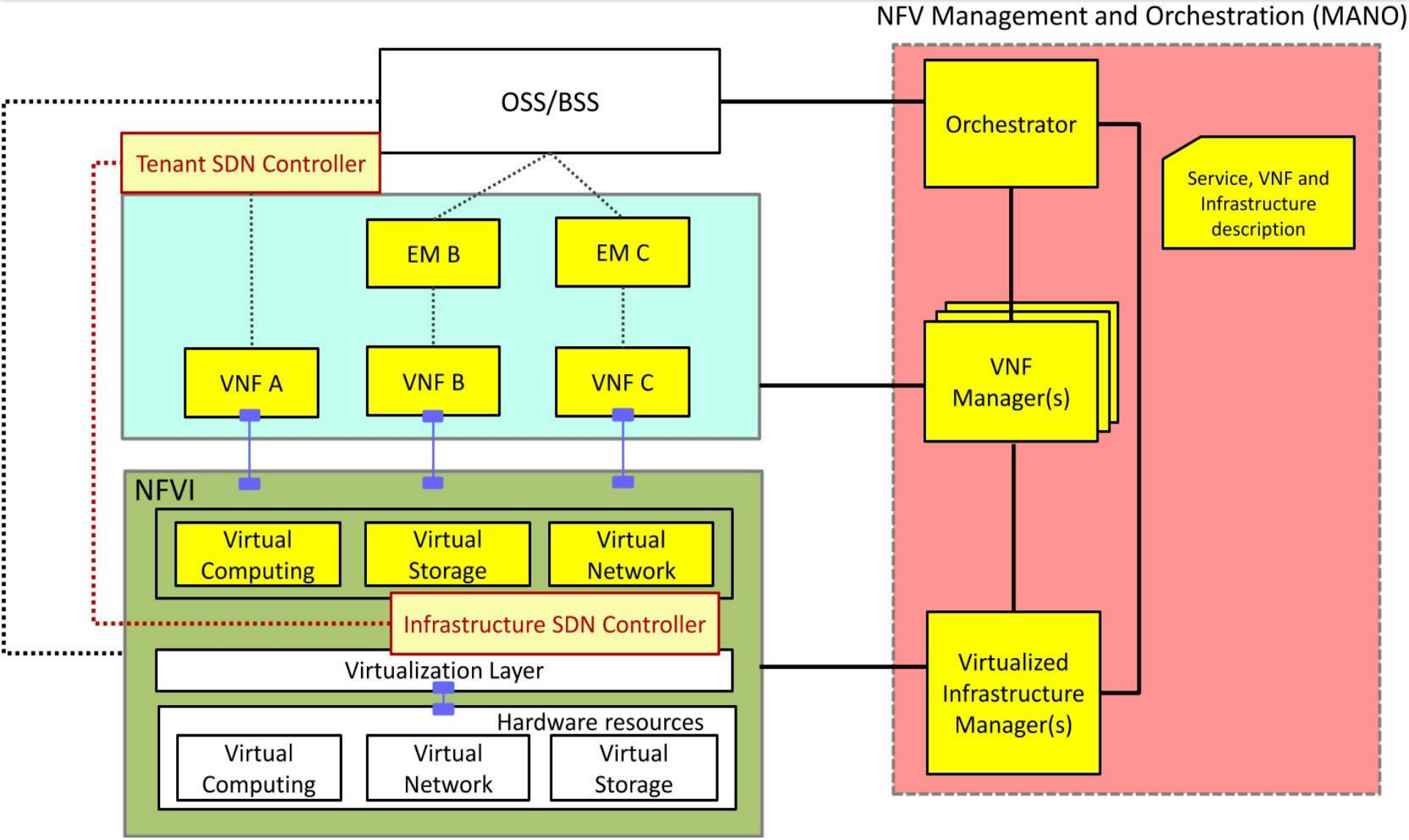
- User plane Function (UPF)
- Data network (DN), e.g. operator services, Internet access or 3rd party services
- Core Access and Mobility Management Function (AMF)
- Authentication Server Function (AUSF)
- Session Management Function (SMF)
- Network Slice Selection Function (NSSF)
- Network Exposure Function (NEF)
- NF Repository Function (NRF)
- Policy Control function (PCF)
- Unified Data Management (UDM)
- Application Function (AF)

5G Architecture Diagram



- User Equipment (UE) like 5G smartphones or 5G cellular devices connect over the 5G New Radio Access Network to the 5G core and further to Data Networks (DN), like the Internet.
- The Access and Mobility Management Function (AMF) acts as a single-entry point for the UE connection.
- Based on the service requested by the UE, the AMF selects the respective Session Management Function (SMF) for managing the user session.
- The User Plane Function (UPF) transports the IP data traffic (user plane) between the User Equipment (UE) and the external networks.
- The Authentication Server Function (AUSF) allows the AMF to authenticate the UE and access services of the 5G core.
- Other functions like the Session Management Function (SMF), the Policy Control Function (PCF), the Application Function (AF) and the Unified Data Management (UDM) function provide the policy control framework, applying policy decisions and accessing subscription information, to govern the network behavior.

SDN and NFV Architecture



5G Network architecture

- NG1: Between UE and AMF (Access and Mobility Management Function)
- NG2: Between RAN (Radio Access Network) or gNB (i.e. 5G base station) and AMF
- NG3: Between RAN or gNB (i.e. 5G base station) and UPF (User Plane Function)
- NG4: Between SMF (Session Management Function) and UPF
- NG5: Between PCF (Policy Control Function) and AF (Application Function).
- NG6: Between UPF and DN (Data Network)
- NG7: NG7 is reference point between SMF and PCF
  NG7r is reference point between vPCF and hPCF
- NG8: Between Unified Data Management (UDM) and AMF
- NG9: Between two core UPFs
- NG10: Reference point between UDM and SMF
- NG11: Between SMF and SMF
- NG12: Between AMF and AUSF (Authentication Server Function)
- NG13: Between UDM and AUSF
- NG14: Between two AMFs
- NG15: • Between PCF and AMF (in Non-roaming scenario)
  • Between V-PCF and AMF (in Roaming scenario)
- NG16: Between two SMFs (Roaming between V-SMF and H-SMF)

Functional blocks within 5G network architecture

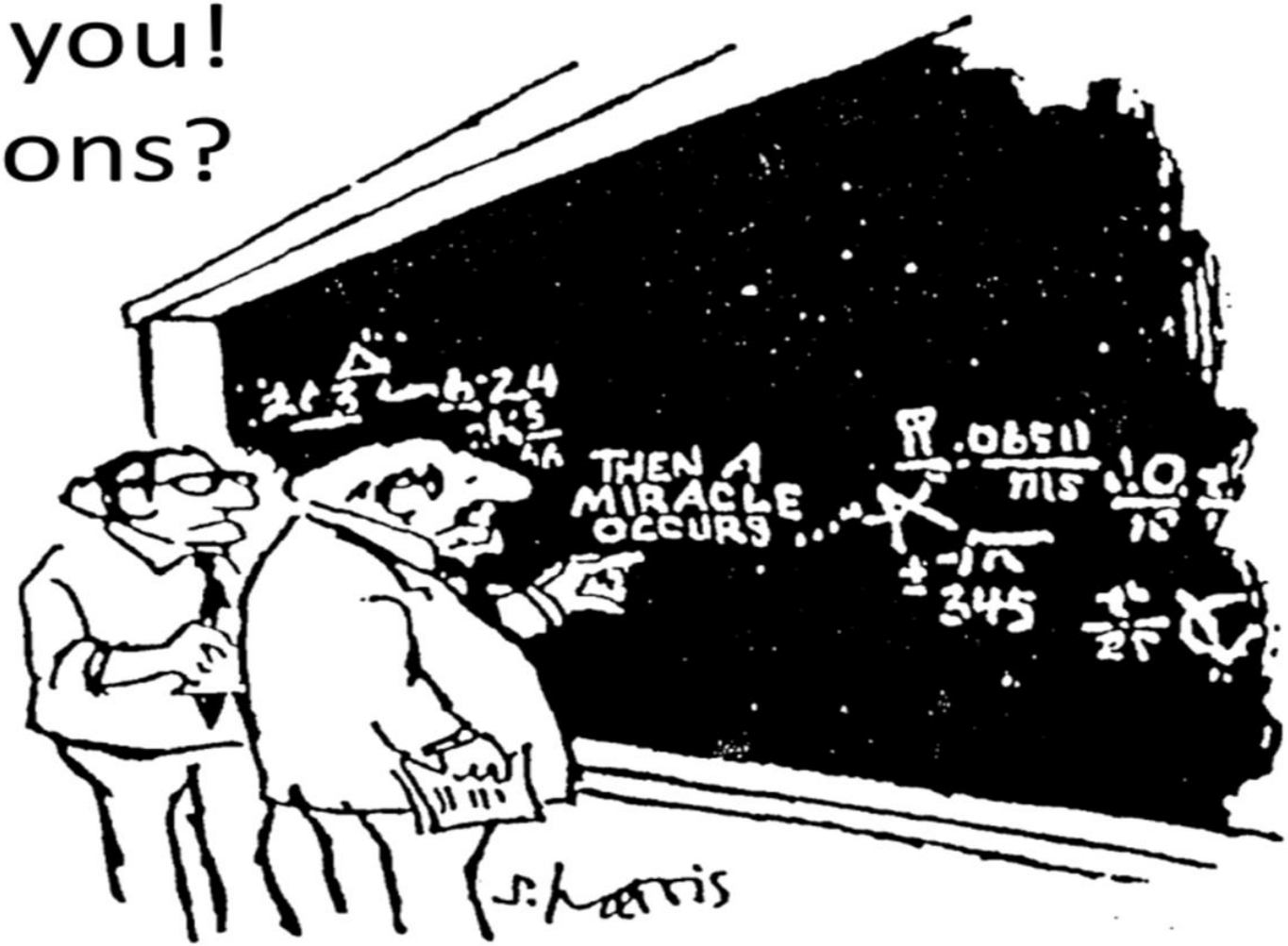
The components of the 5G core architecture include:

- User plane Function (UPF)
- Data network (DN), e.g. operator services, Internet access or 3rd party services
- Core Access and Mobility Management Function (AMF)
- Authentication Server Function (AUSF)
- Session Management Function (SMF)
- Network Slice Selection Function (NSSF)
- Network Exposure Function (NEF)
- NF Repository Function (NRF)
- Policy Control function (PCF)
- Unified Data Management (UDM)
- Application Function (AF)

Acknowledgements and references

We acknowledge the several sources of resources from internet used in presentation meant for seminar only and academic purposes for motivating students and academic community in topic.

Thank you!
Questions?



ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220832059

Black-box and Gray-box Strategies for Virtual Machine Migration.

Conference Paper · January 2007

Source: DBLP

CITATIONS
764

READS
1,378

4 authors:

Timothy Wood
George Washington University
82 PUBLICATIONS **5,843** CITATIONS

SEE PROFILE

Prashant Shenoy
University of Massachusetts Amherst
431 PUBLICATIONS **16,324** CITATIONS

SEE PROFILE

Arun Venkataramani
University of Massachusetts Amherst
79 PUBLICATIONS **6,997** CITATIONS

SEE PROFILE

Mazin Yousif
T-Systems International GmbH
60 PUBLICATIONS **2,913** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Swarming Systems Modeling and Analysis View project

Project self protection of networks View project

All content following this page was uploaded by Timothy Wood on 20 May 2014.

The user has requested enhancement of the downloaded file.

Black-box and Gray-box Strategies for Virtual Machine Migration

Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif[†]
Univ. of Massachusetts Amherst [†]*Intel, Portland*

Abstract

Virtualization can provide significant benefits in data centers by enabling virtual machine migration to eliminate hotspots. We present Sandpiper, a system that automates the task of monitoring and detecting hotspots, determining a new mapping of physical to virtual resources and initiating the necessary migrations. Sandpiper implements a black-box approach that is fully OS- and application-agnostic and a gray-box approach that exploits OS- and application-level statistics. We implement our techniques in Xen and conduct a detailed evaluation using a mix of CPU, network and memory-intensive applications. Our results show that Sandpiper is able to resolve single server hotspots within 20 seconds and scales well to larger, data center environments. We also show that the gray-box approach can help Sandpiper make more informed decisions, particularly in response to memory pressure.

1 Introduction

Data centers—server farms that run networked applications—have become popular in a variety of domains such as web hosting, enterprise systems, and e-commerce sites. Server resources in a data center are multiplexed across multiple applications—each server runs one or more applications and application components may be distributed across multiple servers. Further, each application sees dynamic workload fluctuations caused by incremental growth, time-of-day effects, and flash crowds [1]. Since applications need to operate above a certain performance level specified in terms of a *service level agreement (SLA)*, effective management of data center resources while meeting SLAs is a complex task.

One possible approach for reducing management complexity is to employ *virtualization*. In this approach, applications run on virtual servers that are constructed using virtual machines, and one or more virtual servers are mapped onto each physical server in the system. Virtualization of data center resources provides numerous benefits. It enables application isolation since malicious or greedy applications can not impact other applications co-located on the same physical server. It enables server consolidation and provides better multiplexing of data center resources across applications. Perhaps the biggest advantage of employing virtualization is the ability to flexibly remap physical resources to virtual servers in order to handle workload dynamics. A workload increase can be handled by increasing the resources allocated to a virtual server, if idle resources are available on the physical server, or by simply migrating the virtual server to a less loaded physical server. Migration is transparent to the applications and all modern virtual machines support this capability [6, 15]. However, detecting workload hotspots and initiating a migration is currently handled manually. Manually-initiated migration lacks the agility to respond to sudden workload changes; it is also error-prone since each reshuffle might require migrations or swaps of multiple virtual servers to rebalance system load. Migration is further complicated by the need to consider multiple resources—CPU, network, and memory—for each application and physical server.

To address this challenge, this paper studies automated black-box and gray-box strategies for virtual machine migration in large data centers. Our techniques automate the tasks of monitoring system resource usage, hotspot detection, determining a new mapping and initiating the necessary migrations. More importantly, our black-box techniques can make these decisions by simply observing each virtual machine from the outside and without any knowledge of the application resident within each VM. We also present a gray-box approach that assumes access to a small amount of OS-level statistics in addition to external observations to better inform the migration algorithm. Since a black-box approach is more general by virtue of being OS and application-agnostic, an important aspect of our research is to understand if a black-box approach alone is sufficient and effective for hotspot

detection and mitigation. We have designed and implemented the Sandpiper system to support either black-box, gray-box, or combined techniques. We seek to identify specific limitations of the black-box approach and understand how a gray-box approach can address them.

Sandpiper implements a hotspot detection algorithm that determines *when to migrate* virtual machines, and a hotspot mitigation algorithm that determines *what and where to migrate* and *how much to allocate after the migration*. The hotspot detection component employs a monitoring and profiling engine that gathers usage statistics on various virtual and physical servers and constructs profiles of resource usage. These profiles are used in conjunction with prediction techniques to detect hotspots in the system. Upon detection, Sandpiper's migration manager is invoked for hotspot mitigation. The migration manager employs provisioning techniques to determine the resource needs of overloaded VMs and uses a greedy algorithm to determine a sequence of moves or swaps to migrate overloaded VMs to underloaded servers.

We have implemented our techniques using the Xen virtual machine [3]. We conduct a detailed experimental evaluation on a testbed of two dozen servers using a mix of CPU-, network- and memory-intensive applications. Our results show that Sandpiper can alleviate single server hotspots in less than 20s and more complex multi-server hotspots in a few minutes. Our results show that Sandpiper imposes negligible overheads and that gray-box statistics enable Sandpiper to make better migration decisions when alleviating memory hotspots.

The rest of this paper is structured as follows. Section 2 presents some background, and Sections 3-6 present our design of Sandpiper. Section 7 presents our implementation and evaluation. Finally, Sections 8 and 9 present related work and our conclusions, respectively.

2 Background and System Overview

Existing approaches to dynamic provisioning have either focused on dynamic *replication*, where the number of servers allocated to an application is varied, or dynamic *slicing*, where the fraction of a server allocated to an application is varied; none have considered application *migration* as an option for dynamic provisioning, primarily since migration is not a feasible option in the absence of virtualization. Since migration is transparent to applications executing within virtual machines, our work considers this third approach—resource provisioning via dynamic migrations in virtualized data centers. We present *Sandpiper*[1], a system for automated migration of virtual servers in a data center to meet application SLAs. Sandpiper assumes a large cluster of possibly heterogeneous servers. The hardware configuration of each server—its CPU, network interface, disk and memory characteristics—is assumed to be known to Sandpiper. Each physical server (also referred to as a physical machine or PM) runs a *virtual machine monitor* and one or more virtual machines. Each virtual server runs an application or an application component (the terms virtual servers and virtual machine are used interchangeably). Sandpiper currently uses Xen to implement such an architecture. Each virtual server is assumed to be allocated a certain slice of the physical server resources. In the case of CPU, this is achieved by assigning a weight to the virtual server and the underlying Xen CPU scheduler allocates CPU bandwidth in proportion to the weight. In case of the network interface, Xen is yet to implement a similar fair-share scheduler; a best-effort FIFO scheduler is currently used and Sandpiper is designed to work with this constraint. In case of memory, a slice is assigned by allocating a certain amount of RAM to each resident VM. All storage is assumed to be on a network file system or a storage area network, thereby eliminating the need to move disk state during VM migrations [6].

[1] A migratory bird.

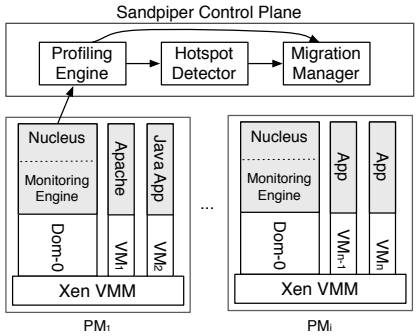


Figure 1: The Sandpiper architecture.

Sandpiper runs a component called the *nucleus* on each physical server; the nucleus runs inside a special virtual server (domain 0 in Xen) and is responsible for gathering resource usage statistics on that server (see Figure 1). It employs a *monitoring engine* that gathers processor, network interface and memory swap statistics for each virtual server. For gray-box approaches, it implements a daemon within each virtual server to gather OS-level statistics and perhaps application logs.

The nuclei periodically relay these statistics to the Sandpiper *control plane*. The control plane runs on a distinguished node and implements much of the intelligence in Sandpiper. It comprises three components: a *profiling engine*, a *hotspot detector* and a *migration manager* (see Figure 1). The profiling engine uses the statistics from the nuclei to construct resource usage profiles for each virtual server and aggregate profiles for each physical server. The hotspot detector continuously monitors

these usage profiles to detect hotspots —informally, a hotspot is said to have occurred if the aggregate usage of any resource (processor, network or memory) exceeds a threshold or if SLA violations occur for a "sustained" period. Thus, the hotspot detection component determines *when* to signal the need for migrations and invokes the migration manager upon hotspot detection, which attempts hotspot mitigation via dynamic migrations. It implements algorithms that determine *what* virtual servers to migrate from the overloaded servers, *where* to move them, and *how much* of a resource to allocate the virtual servers once the migration is complete (i.e., determine a new resource allocation to meet the target SLAs). The migration manager assumes that the virtual machine monitor implements a migration mechanism that is transparent to applications and uses this mechanism to automate migration decisions; Sandpiper currently uses Xen's migration mechanisms that were presented in [6].

3 Monitoring and Profiling in Sandpiper

This section discusses online monitoring and profile generation in Sandpiper.

3.1 Unobtrusive Black-box Monitoring

The monitoring engine is responsible for tracking the processor, network and memory usage of each virtual server. It also tracks the total resource usage on each physical server by aggregating the usages of resident VMs. The monitoring engine tracks the usage of each resource over a measurement interval $\mathcal{I}$ and reports these statistics to the control plane at the end of each interval.

In a pure black-box approach, all usages must be inferred solely from external observations and without relying on OS-level support inside the VM. Fortunately, much of the required information can be determined directly from the Xen hypervisor or by monitoring events within domain-0 of Xen. Domain-0 is a distinguished VM in Xen that is responsible for I/O processing; domain-0 can host device drivers and act as a "driver" domain that processes I/O requests from other domains [3, 9]. As a result, it is possible to track network and disk I/O activity of various VMs by observing the driver activity in domain-0 [9]. Similarly, since CPU scheduling is implemented in the Xen hypervisor, the CPU usage of various VMs can be determined by tracking scheduling events in the hypervisor [10]. Thus, black-box monitoring can be implemented in the nucleus by tracking various domain-0 events and without modifying any virtual server. Next, we discuss CPU, network and memory monitoring using this approach.

CPU Monitoring: By instrumenting the Xen hypervisor, it is possible to provide domain-0 with access to CPU scheduling events which indicate when a VM is scheduled and when it relinquishes the CPU. These events are tracked to determine the duration for which each virtual machine is scheduled within each measurement interval $\mathcal{I}$. The Xen 3.0 distribution includes a monitoring application called *XenMon* [10] that tracks the CPU usages of the resident virtual machines using this approach; for simplicity, the monitoring engine employs a modified version of XenMon to gather CPU usages of resident VMs over a configurable measurement interval $\mathcal{I}$.

It is important to realize that these statistics do not capture the CPU overhead incurred for processing disk and network I/O requests; since Xen uses domain-0 to process disk and network I/O requests on behalf of other virtual machines, this processing overhead gets charged to the CPU utilization of domain 0. To properly account for this request processing ovehead, analogous to proper accounting of interrupt processing overhead in OS kernels, we must apportion the CPU utilization of domain-0 to other virtual machines. We assume that the monitoring engine and the nucleus impose negligible overhead and that all of the CPU usage of domain-0 is primarily due to requests processed on behalf of other VMs. Since domain-0 can also track I/O request events based on the number of memory page exchanges between domains, we determine the number of disk and network I/O requests that are processed for each VM. Each VM is then charged a fraction of domain-0's usage based on the proportion of the total I/O requests made by that VM. A more precise approach requiring a modified scheduler was proposed in [9].

Network Monitoring: Domain-0 in Xen implements the network interface driver and all other domains access the driver via clean device abstractions. Xen uses a virtual firewall-router (VFR) interface; each domain attaches one or more virtual interfaces to the VFR [3]. Doing so enables Xen to multiplex all its virtual interfaces onto the underlying physical network interface.

Consequently, the monitoring engine can conveniently monitor each VM's network usage in Domain-0. Since each virtual interface looks like a modern NIC and Xen uses Linux drivers, the monitoring engines can use the Linux `/proc` interface (in particular `/proc/net/dev`) to monitor the number of bytes sent and received on each interface. These statistics are gathered over interval $\mathcal{I}$ and returned to the control plane.

Memory Monitoring: Black-box monitoring of memory is challenging since Xen allocates a user-specified amount of memory to each VM and requires the OS within the VM to manage that memory; as a result, the memory utilization is only known to the OS within each VM. It is possible to instrument Xen to observe memory accesses within each VM through the use of shadow page tables, which is used by Xen's migra-

tion mechanism to determine which pages are dirtied during migration. However, trapping each memory access results in a significant application slowdown and is only enabled during migrations[6]. Thus, memory usage statistics are not directly available and must be inferred.

The only behavior that is visible externally is *swap activity*. Since swap partitions reside on a network disk, I/O requests to swap partitions need to be processed by domain-0 and can be tracked. By tracking the reads and writes to each swap partition from domain-0, it is possible to detect memory pressure within each VM. The recently proposed Geiger system has shown that such passive observation of swap activity can be used to infer useful information about the virtual memory subsystem such as working set sizes [11].

Our monitoring engine tracks the number of read and write requests to swap partitions within each measurement interval $\mathcal{I}$ and reports it to the control plane. Since substantial swapping activity is indicative of memory pressure, our current black-box approach is limited to reactive decision making and can not be proactive.

3.2 Gray-box Monitoring

Black-box monitoring is useful in scenarios where it is not feasible to "peek inside" a VM to gather usage statistics. Hosting environments, for instance, run third-party applications, and in some cases, third-party installed OS distributions. Amazon's Elastic Computing Cloud (EC2) service, for instance, provides a "barebone" virtual server where customers can load their own OS images. While OS instrumentation is not feasible in such environments, there are environments such as corporate data centers where both the hardware infrastructure and the applications are owned by the same entity. In such scenarios, it is feasible to gather OS-level statistics as well as application logs, which can potentially enhance the quality of decision making in Sandpiper.

Sandpiper supports gray-box monitoring, when feasible, using a light-weight monitoring daemon that is installed inside each virtual server. In Linux, the monitoring daemon uses the `/proc` interface to gather OS-level statistics of CPU, network, and memory usage. The memory usage monitoring, in particular, enables proactive detection and mitigation of memory hotspots. The monitoring daemon also can process logs of applications such as web and database servers to derive statistics such as request rate, request drops and service times. Direct monitoring of such application-level statistics enables explicit detection of SLA violations, in contrast to the black-box approach that uses resource utilizations as a proxy metric for SLA monitoring.

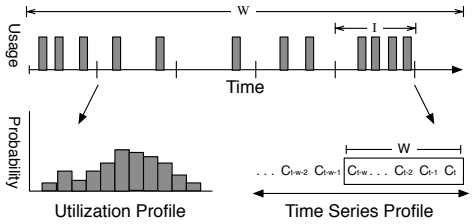


Figure 2: Profile generation in Sandpiper

3.3 Profile Generation

The profiling engine receives periodic reports of resource usage from each nucleus. It maintains a usage history for each server, which is then used to compute a profile for each virtual and physical server. A profile is a compact description of that server's resouce usage over a sliding time window W. Three black-box profiles are maintained per virtual server: CPU utilization, network bandwidth utilization, and swap rate (i.e., page fault rate). If gray-box monitoring is permitted, four additional profiles are maintained: memory utilization, service time, request drop rate and incoming request rate. Similar profiles are also maintained for each physical server, which indicate the aggregate usage of resident VMs.

Each profile contains a distribution and a time series. The distribution, also referred to as the distribution profile, represents the probability distribution of the resource usage over the window W. To compute a CPU distribution profile, for instance, a histogram of observed usages over all intervals $\mathcal{I}$ contained within the window W is computed; normalizing this histogram yields the desired probability distribution (see Figure 2).

While a distribution profile captures the variations in the resource usage, it does not capture temporal correlations. For instance, a distribution does not indicate whether the resource utilization increased or decreased within the window W. A time-series profile captures these temporal fluctuations and is is simply a list of all reported observations within the window W. For instance, the CPU time-series profile is a list $(C_1, C_2, ..., C_k)$ of the k reported utilizations within the window W. Whereas time-series profiles are used by the hotspot detector to spot increasing utilization trends, distribution profiles are used by the migration manager to estimate peak resource requirements and provision accordingly.

4 Hotspot Detection

The hotspot detection algorithm is responsible for signaling a need for VM migration whenever SLA violations are detected implicitly by the black-box approach or explicitly by the gray-box approach. Hotspot detec-

tion is performed on a per-physical server basis in the back-box approach—a hot-spot is flagged if the aggregate CPU or network utilizations on the physical server exceed a threshold or if the total swap activity exceeds a threshold. In contrast, explicit SLA violations must be detected on a per-virtual server basis in the gray-box approach—a hotspot is flagged if the memory utilization of the VM exceeds a threshold or if the response time or the request drop rate exceed the SLA-specified values.

To ensure that a small transient spike does not trigger needless migrations, a hotspot is flagged only if thresholds or SLAs are exceeded for a sustained time. Given a time-series profile, a hotspot is flagged if at least k out the n most recent observations as well as the next predicted value exceed a threshold. With this constraint, we can filter out transient spikes and avoid needless migrations. The values of k and n can be chosen to make hotspot detection aggressive or conservative. For a given n, small values of k cause aggressive hotspot detection, while large values of k imply a need for more sustained threshold violations and thus a more conservative approach. Similarly, larger values of n incorporate a longer history, resulting in a more conservative approach. In the extreme, $n = k = 1$ is the most aggressive approach that flags a hostpot as soon as the threshold is exceeded. Finally, the threshold itself also determines how aggressively hotspots are flagged; lower thresholds imply more aggressive migrations at the expense of lower server utilizations, while higher thresholds imply higher utilizations with the risk of potentially higher SLA violations.

In addition to requiring k out of n violations, we also require that the next predicted value exceed the threshold. The additional requirement ensures that the hotspot is likely to persist in the future based on current observed trends. Also, predictions capture rising trends, while preventing declining ones from triggering a migration.

Sandpiper employs time-series prediction techniques to predict future values [4]. Specifically, Sandpiper relies on the auto-regressive family of predictors, where the n-th order predictor $AR(n)$ uses n prior observations in conjunction with other statistics of the time series to make a prediction. To illustrate the first-order AR(1) predictor, consider a sequence of observations: u_1, u_2, ..., u_k. Given this time series, we wish to predict the demand in the $(k+1)$th interval. Then the first-order *AR(1) predictor* makes a prediction using the previous value u_k, the mean of the the time series values μ, and the parameter ϕ which captures the variations in the time series [4]. The prediction $\hat{u}_{k+1}$ is given by:

$$\hat{u}_{k+1} = \mu + \phi(u_k - \mu) \quad (1)$$

As new observations arrive from the nuclei, the hot spot detector updates its predictions and performs the above checks to flag new hotspots in the system.

5 Resource Provisioning

A hotspot indicates a resource deficit on the underlying physical server to service the collective workloads of resident VMs. Before the hotspot can be resolved through migrations, Sandpiper must first estimate *how much* additional resources are needed by the overloaded VMs to fulfill their SLAs; these estimates are then used to locate servers that have sufficient idle resources.

5.1 Black-box Provisioning

The provisioning component needs to estimate the *peak* CPU, network and memory requirement of each overloaded VM; doing so ensures that the SLAs are not violated even in the presence of peak workloads.

Estimating peak CPU and network bandwidth needs: Distribution profiles are used to estimate the peak CPU and network bandwidth needs of each VM. The tail of the usage distribution represents the peak usage over the recent past and is used as an estimate of future peak needs. This is achieved by computing a high percentile (e.g., the 95th percentile) of the CPU and network bandwidth distribution as an initial estimate of the peak needs.

Since both the CPU scheduler and the network packet scheduler in Xen are work-conserving, a VM can use more than its fair share, provided that other VMs are not using their full allocations. In case of the CPU, for instance, a VM can use a share that exceeds the share determined by its weight, so long as other VMs are using less than their weighted share. In such instances, the tail of the distribution will exceed the guaranteed share and provide insights into the actual peak needs of the application. Hence, a high percentile of the distribution is a good first approximation of the peak needs.

However, if *all* VMs are using their fair shares, then an overloaded VM will not be allocated a share that exceeds its guaranteed allocation, even though its peak needs are higher than the fair share. In such cases, the observed peak usage (i.e., the tail of the distribution) will equal its fair-share. In this case, the tail of the distribution will *under-estimate* the actual peak need. To correct for this under-estimate, the provisioning component must scale the observed peak to better estimate the actual peak. Thus, whenever the CPU or the network interface on the physical server are close to saturation, the provisioning component first computes a high-percentile of the observed distribution and then adds a constant Δ to scale up this estimate.

Example *Consider two virtual machines that are assigned CPU weights of 1:1 resulting in a fair share of 50% each. Assume that VM_1 is overloaded and requires 70% of the CPU to meet its peak needs. If VM_2 is underloaded and only using 20% of the CPU, then the work-*

conserving Xen scheduler will allocate 70% to VM_1. In this case, the tail of the observed distribution is a good inddicator of VM_1's peak need. In contrast, if VM_2 is using its entire fair share of 50%, then VM_1 will be allocated exactly its fair share. In this case, the peak observed usage will be 50%, an underestimate of the actual peak need. Since Sandpiper can detect that the CPU is fully utilized, it will estimate the peak to be $50 + \Delta$.

The above example illustrates a fundamental limitation of the black-box approach—it is not possible to estimate the true peak need when the underlying resource is fully utilized. The scale-up factor Δ is simply a guess and might end up over- or under-estimating the true peak.

Estimating peak memory needs: Xen allows a fixed amount of physical memory to be assigned to each resident VM; this allocation represents a hard upper-bound that can not be exceeded regardless of memory demand and regardless of the memory usage in other VMs. Consequently, our techniques for estimating the peak CPU and network usage do not apply to memory. The provisioning component uses observed swap activity to determine if the current memory allocation of the VM should be increased. If swap activity exceeds the threshold indicating memory pressure, then the the current allocation is deemed insufficient and is increased by a constant amount Δ_m. Observe that techniques such as Geiger that attempt to infer working set sizes by observing swap activity [11] can be employed to obtain a better estimate of memory needs; however, our current prototype uses the simpler approach of increasing the allocation by a fixed amount Δ_m whenever memory pressure is observed.

5.2 Gray-box Provisioning

Since the gray-box approach has access to application-level logs, information contained in the logs can be utilized to estimate the peak resource needs of the application. Unlike the black-box approach, the peak needs can be estimated even when the resource is fully utilized.

To estimate peak needs, the peak request arrival rate is first estimated. Since the number of serviced requests as well as the the number of dropped requests are typically logged, the incoming request rate is the summation of these two quantities. Given the distribution profile of the arrival rate, the peak rate is simply a high percentile of the distribution. Let λ_{peak} denote the estimated peak arrival rate for the application.

Estimating peak CPU needs: An application model is necessary to estimate the peak CPU needs. Applications such as web and database servers can be modeled as G/G/1 queuing systems [23]. The behavior of such a G/G/1 queuing system can be captured using the following queuing theory result [13]:

$$\lambda_{cap} \geq \left[s + \frac{\sigma_a^2 + \sigma_b^2}{2 \cdot (d - s)} \right]^{-1} \quad (2)$$

where d is the mean response time of requests, s is the mean service time, and λ_{cap} is the request arrival rate. σ_a^2 and σ_b^2 are the variance of inter-arrival time and the variance of service time, respectively. Note that response time includes the full queueing delay, while service time only reflects the time spent actively processing a request.

While the desired response time d is specified by the SLA, the service time s of requests as well as the variance of inter-arrival and service times σ_a^2 and σ_b^2 can be determined from the server logs. By substituting these values into Equation 2, a lower bound on request rate λ_{cap} that can be serviced by the virtual server is obtained. Thus, λ_{cap} represents the current capacity of the VM.

To service the estimated peak workload λ_{peak}, the current CPU capacity needs to be scaled by the factor $\frac{\lambda_{peak}}{\lambda_{cap}}$. Observe that this factor will be greater than 1 if the peak arrival rate exceeds the currently provisioned capacity. Thus, if the VM is currently assigned a CPU weight w, its allocated share needs to be scaled up by the factor $\frac{\lambda_{peak}}{\lambda_{cap}}$ to service the peak workload.

Estimating peak network needs: The peak network bandwidth usage is simply estimated as the product of the estimated peak arrival rate λ_{peak} and the mean requested file size b; this is the amount of data transferred over the network to service the peak workload. The mean request size can be computed from the server logs.

6 Hotspot Mitigation

Once a hotspot has been detected and new allocations have been determined for overloaded VMs, the migration manager invokes its hotspot mitigation algorithm. This algorithm determines *which* virtual servers to migrate and *where* in order to dissipate the hotspot. Determining a new mapping of VMs to physical servers that avoids threshold violations is NP-hard—the multi-dimensional bin packing problem can be reduced to this problem, where each physical server is a bin with dimensions corresponding to its resource constraints and each VM is an object that needs to be packed with size equal to its resource requirements. Even the problem of determining if a valid packing exists is NP-hard.

Consequently, our hotspot mitigation algorithm resorts to a heuristic to determine which overloaded VMs to migrate and where *such that migration overhead is minimized.* Reducing the migration overhead (i.e., the amount of data transferred) is important, since Xen's live migration mechanism works by iteratively copying the

memory image of the VM to the destination while keeping track of which pages are being dirtied and need to be resent. This requires Xen to intercept all memory accesses for the migrating domain, which significantly impacts the performance of the application inside the VM. By reducing the amount of data copied over the network, Sandpiper can minimize the total migration time, and thus, the performance impact on applications. Note that network bandwidth available for application use is also reduced due to the background copying during migrations; however, on a gigabit LAN, this impact is small.

Capturing Multi-dimensional Loads: Once the desired resource allocations have been determined by either our black-box or gray-box approach, the problem of finding servers with sufficient idle resource to house overloaded VMs is identical for both. The migration manager employs a greedy heuristic to determine which VMs need to be migrated. The basic idea is to move load from the most overloaded servers to the least-overloaded servers, while attempting to minimize data copying incurred during migration. Since a VM or a server can be overloaded along one or more of three dimensions–CPU, network and memory–we define a new metric that captures the combined CPU-network-memory load of a virtual and physical server. The *volume* of a physical or virtual server is defined as the product of its CPU, network and memory loads:

$$Vol = \frac{1}{1 - cpu} * \frac{1}{1 - net} * \frac{1}{1 - mem} \qquad (3)$$

where cpu, net and mem are the corresponding utilizations of that resource for the virtual or physical server.[2] The higher the utilization of a resource, the greater the volume; if multiple resources are heavily utilized, the above product results in a correspondingly higher volume. The volume captures the degree of (over)load along multiple dimensions in a unified fashion and can be used by the mitigation algorithms to handle all resource hotspots in an identical manner.

Migration Phase: To determine which VMs to migrate, the algorithm orders physical servers in decreasing order of their volumes. Within each server, VMs are considered in decreasing order of their *volume-to-size ratio (VSR)*; where VSR is defined as *Volume*/*Size*; size is the memory footprint of the VM. By considering VMs in VSR order, the algorithm attempts to migrate the maximum volume (i.e., load) per unit byte moved, which has been shown to minimize migration overhead [20].

[2]If a resource is fully utilized, its utilization is set to $1 - \epsilon$, rather than one, to avoid infinite volume servers. Also, since the black-box approach is oblivious of the precise memory utilization, the value of mem is set to 0.5 in the absence of swapping and to $1 - \epsilon$ if memory pressure is observed; the precise value of mem is used in the gray-box approach.

The algorithm proceeds by considering the highest VSR virtual machine from the highest volume server and determines if it can be housed on the least volume (least loaded) physical server. The move is feasible only if that server has sufficient idle CPU, network and memory resources to meet the desired resource allocation of the candidate VM as determined by the provisioning component (Section 5). Since we use VSR to represent three resource quantities, the least loaded server may not necessarily "fit" best with a particular VM's needs. If sufficient resources are not available, then the algorithm examines the next least loaded server and so on, until a match is found for the candidate VM. If no physical server can house the highest VSR VM, then the algorithm moves on to the next highest VSR VM and attempts to move it in a similar fashion. The process repeats until the utilizations of all resources on the physical server fall below their thresholds.

The algorithm then considers the next most loaded physical server that is experiencing a hotspot and repeats the process until there are no physcial servers left with a hotspot. The output of this algorithm is a list of overloaded VMs and a new destination server for each; the actual migrations are triggered only after all moves have been determined.

Swap Phase: In cases where there aren't sufficient idle resources on less loaded servers to dissipate a hotspot, the migration algorithm considers VM swaps as an alternative. A swap involves exchanging a high VSR virtual machine from a loaded server with one or more low VSR VMs from an underloaded server. Such a swap reduces the overall utilization of the overloaded server, albeit to a lesser extent than a one-way move of the VM. Our algorithm considers the highest VSR VM on the highest volume server with a hotspot; it then considers the lowest volume server and considers the k lowest VSR VMs such that these VMs collectively free up sufficient resources to house the overloaded VM. The swap is considered feasible if the two physical servers have sufficient resources to house the other server's candidate VM(s) without violating utilization thresholds. If a swap cannot be found, the next least loaded server is considered for a possible swap and so on. The process repeats until sufficient high VSR VMs have been swapped with less loaded VMs so that the hotspot is dissipated. Although multi-way swaps involving more than two servers can also be considered, our algorithm presently does not implement such complex swaps. The actual migrations to perform the swaps are triggered only after a list of all swaps is constructed. Note that a swap may require a third server with "scratch" RAM to temporarily house a VM before it moves to its final destination. An alternative is to (i) suspend one of the VMs on disk, (ii) use the freed up RAM to accommodate the other VM, and (iii)

VM	Peak 1	Peak 2	Peak 3	RAM (MB)	Start PM
1	200	130	130	256	1
2	90	90	90	256	1
3	60	200	60	256	2
4	60	90	90	256	2
5	10	10	130	128	3

Table 1: Workload in requests/second, memory allocations, and initial placement.

resume the first VM on the other server; doing so is not transparent to the temporarily suspended VM.

7 Implementation and Evaluation

The implementation of Sandpiper is based on Xen. The Sandpiper *control plane* is implemented as a daemon that runs on the control node. It listens for periodic usage reports from the various nuclei, which are used to generate profiles. The profiling engine currently uses a history of the past 200 measurements to generate virtual and physical server profiles. The hotspot detector uses these profiles to detect hotspots; currently a hotspot is triggered when 3 out of 5 past readings and the next predicted value exceeds a threshold. The default threshold is set to 75%. The migration manager implements our provisioning and hotspot mitigation algorithms; it notifies the nuclei of any desired migrations, which then trigger them. In all, the control plane consists of less than 750 lines of python code.

The Sandpiper *nucleus* is a Python application that extends the XenMon CPU monitor to also acquire network and memory statistics for each VM. The monitoring engine in the nucleus collects and reports measurements once every 10 seconds—the default measurement interval. The nucleus uses Xen's Python management API to trigger migrations and adjust resource allocations as directed by the control plane. While black-box monitoring only requires access to domain-0 events, gray-box monitoring employs two additional components: a Linux OS daemon and an Apache module.

The gray-box linux daemon runs on each VM that permits gray-box monitoring. It currently gathers memory statistics via the `/proc` interface—the memory utilization, the number of free pages and swap usage are reported to the monitoring engine in each interval. The gray-box apache module comprises of a real-time log analyzer and a dispatcher. The log-analyzer processes log-entries as they are written to compute statistics such as the service time, request rate, request drop rate, inter-arrival times, and request/file sizes. The dispatcher is implemented as a kernel module based on Linux IP Virtual server (IPVS) ver 1.2.1; the goal of the kernel module is to accurately estimate the request arrival rate during overload periods, when a high fraction of requests may be dropped. Since requests can be dropped at the TCP layer as well as at the HTTP layer during overloads, the use of a transport-level dispatcher such as IPVS is necessary for accurately estimating the drop (and hence arrival) rates. Ordinarily, the kernel dispatcher simply forwards incoming requests to Apache for processing. In all, the nucleus comprises 650 lines of Python code.

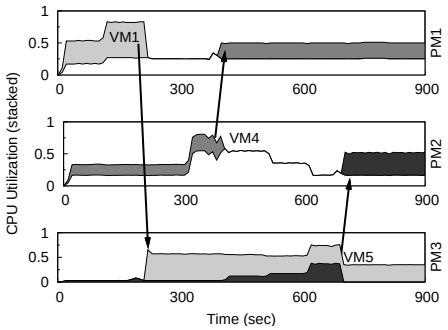


Figure 3: A series of migrations resolve hotspots. Different shades are used for each migrating VM.

Our evaluation of Sandpiper is based on a prototype data center consisting of twenty 2.4Ghz Pentium-4 servers connected over gigabit ethernet. All servers run Linux 2.6.16 and Xen 3.0.2-3 and are equipped with at least 1GB of RAM. A cluster of Pentium-3 Linux servers is used to generate workloads for our experiments. One node in the cluster is designated to run the Sandpiper control plane, while the rest host one or more VMs, all of which run the Sandpiper nucleus in domain- 0. In the following experiments, our VMs run Apache 2.0.54, PHP 4.3.10, and MySQL 4.0.24.

7.1 Migration Effectiveness

Our first experiment demonstrates the effectiveness of migrations and swaps by exercising Sandpiper's hotspot detection and migration algorithms; we subject a set of black-box servers to a series of workloads that repeatedly place the system in overload. Our experiment uses three physical servers and five VMs with memory allocations as shown in Table 1. All VMs run Apache serving dynamic PHP web pages. The PHP scripts are designed to be CPU intensive so that a low client request rate places a large CPU load on a server without significant network or memory utilization. We use *httperf* to inject a workload that goes through three phases, each of which causes a hotspot on a different physical machine. The peak request rates for each phase are shown in Table 1.

Figure 3 presents a time series of the load placed on each VM along with the triggered migrations. In the first phase, a large load is placed on VM_1, causing the CPU utilization on PM_1 to exceed the CPU threshold. The

system detects a hotspot at t=166 seconds. The migration manager examines candidates for migration in VSR order. VM_1 has the highest VSR, so it is selected as a candidate. Since PM_3 has sufficient spare capacity to house VM_1, it is migrated there, thereby eliminating the hotspot. This represents the ideal case for our algorithm: if possible, we try to migrate the most loaded VM from an overloaded PM to one with sufficient spare capacity.

In the second phase, PM_2 becomes overloaded due to increasing load on VM_3. However, the migration manager is unable to migrate this VM because there is insufficient capacity on the other PMs. As a result, at t=362 seconds, the VM on PM_2 with the second highest VSR VM_4, is migrated to PM_1 that now has spare capacity. This demonstrates a more typical case where none of the underloaded PMs have sufficient spare capacity to run the overloaded PM's highest VSR VM, so instead we migrate less overloaded VMs that can fit elsewhere.

In the final phase, PM_3 becomes overloaded when both of its VMs receive *identical* large loads. Unlike the previous two cases where candidate VMs had identical memory footprints, VM_5 has half as much RAM as VM_1, so it is chosen for migration. By selecting the VM with a lower footprint, Sandpiper maximizes the reduction in load per megabyte of data transfered.

Result: To eliminate hotspots while minimzing the overhead of migration, our placement algorithm tries to move the highest VSR VM to the least loaded PM. This maximizes the amount of load displaced from the hotspot per megabyte of data transferred.

7.2 Virtual Machine Swaps

Next we demonstrate how VM swaps can mitigate hotspots. The first two VMs in our setup are allocated 384 MB of RAM on PM_1; VM_3 and VM_4 are assigned 256 MB each on PM_2. The load on VM_1 steadily increases during the experiment, while the others are constant. As before, clients use httperf to request dynamic PHP pages.

Figure 4 shows that a hotspot is detected on PM_1 due to the increasing load on VM_1. However, there is insufficient spare capacity on PM_2 to support a migrated VM. The only viable solution is to swap VM_2 with VM_4. To facilitate such swaps, Sandpiper uses spare RAM on the control node as scratch space.

By utilizing this scratch space, Sandpiper never requires either physical server to simultaneously run both VMs. It does require us to perform three migrations instead of two; however, Sandpiper reduces the migration cost by always migrating the smaller footprint VM via the scratch server. As shown in Figure 4, the load on PM_2 drops at t=219 due to the migration of VM_4 to scratch. VM_2 is then migrated directly from PM_1 to PM_2

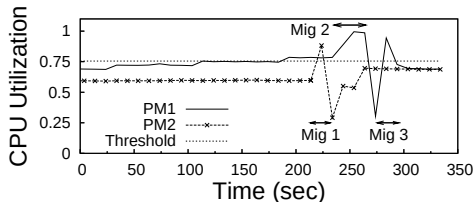
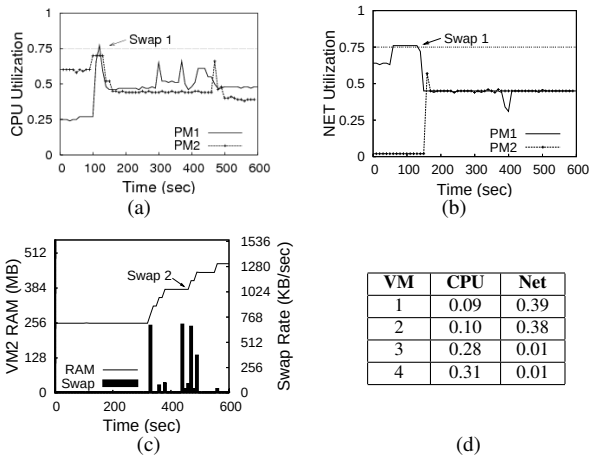


Figure 4: After a hotspot is detected on PM_1, a swap occurs via the scratch PM. Initially, VM_1 and VM_2 are on PM_1. VM_2 is swapped with VM_4 from PM_2 to resolve overload.



VM	CPU	Net
1	0.09	0.39
2	0.10	0.38
3	0.28	0.01
4	0.31	0.01

Figure 5: Swaps and migrations to handle network- and memory-intensive loads. Initially, VM_1 and VM_2 are on PM_1, the rest on PM_2. After two swaps, PM_1 hosts VM_1 and VM_4.

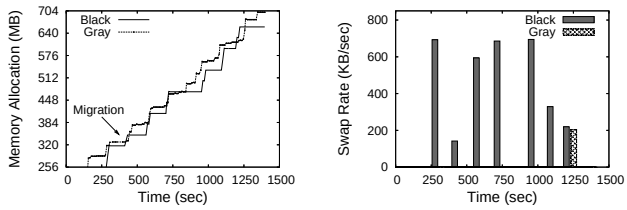
at t=233, followed by a migration of VM_4 from scratch to PM_1, which takes an additional 10s.

The figure also depicts the CPU overhead of a migration—as indicated by a rise in CPU utilization of the initiating server whenever a migration begins. This suggests using lower CPU hotspot thresholds to safely absorb the additional overheads caused by a migration.

Result: Swaps incur more overhead, but increase the chances of mitigating hotspots in clusters with high average utilization.

7.3 Mixed Resource Workloads

Sandpiper can consolidate applications that stress different resources to improve the overall multiplexing of server resources. Our setup comprises two servers with two VMs each. Both VMs on the first server are network-intensive, involving large file transfers, while those on the second server are CPU-intensive running Apache with dynamic PHP scripts. All VMs are initially allocated 256 MB of memory. VM_2 additionally runs a main-memory database that stores its tables in memory, causing its memory usage to grow over time.



(a) (b)

Figure 6: The black-box system lags behind the gray-box system in allocating memory. With its detailed memory statistics, the gray-box approach is proactive and increases memory without waiting until swapping occurs.

Figures 5(a) and (b) show the resource utilization of each PM over time. Since PM_1 has a network hotspot and PM_2 has a CPU hotspot, Sandpiper swaps a network-intensive VM for a CPU-intensive VM at t=130. This results in a lower CPU and network utilization on both servers. Figure 5(d) shows the initial utilizations of each VM; after the swap, the aggregate CPU and network utilizations on both servers falls below 50%.

In the latter half, memory pressure increases on VM_2 due to its main-memory database application. As shown in 5(c), Sandpiper responds by increasing the RAM allocation in steps of 32MB every time swapping is observed; when no additional RAM is available, the VM is swapped to the second physical server at t=430. This is feasible because two cpu-intensive jobs are swapped, leaving CPU and network utilization balanced, and the second physical server has more RAM than the first. Memory allocations are reactive since only black-box stats are available. Next we demonstrate how a gray-box approach can proactively respond to memory pressure.

Result: Sandpiper can respond to network, CPU, or memory hotspots and can collocate VMs that stress different resources to improve overall system utilization.

7.4 Gray v. Black: Memory Allocation

We compare the effectiveness of the black- and gray-box approaches in mitigating memory hotspots using the SPECjbb 2005 benchmark. SPECjbb emulates a three-tier web application based on J2EE servers. We use SPECjbb to apply an increasingly intense workload to a single VM. The workload increases every two minutes, causing a significant increase in memory usage.

The VM is initially assigned 256MB of RAM, and resides on a physical machine with 384 MB total RAM. We also run a second, idle physical server which has 1GB RAM. We run the experiment with two separate pairs of servers, Black and Gray, that correspond to the black- and gray-box approaches, respectively. The Gray system is configured to signal a hotspot whever the amount of free RAM in the virtual machine falls below 32MB.

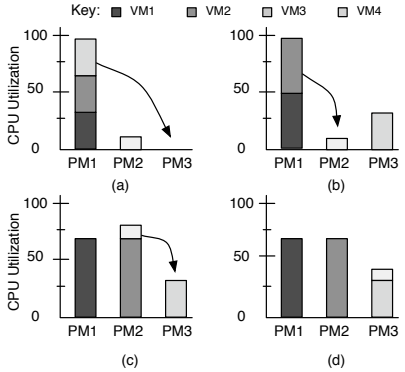
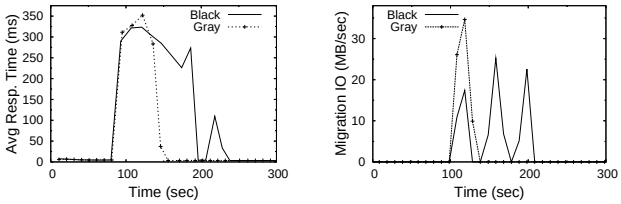


Figure 7: The black-box system incorrectly guesses resource requirements since CPU usage is saturated, resulting in an increased resolution time. The gray-box system infers usage requirements and transitions directly from a) to d).



(a) (b)

Figure 8: The gray-box system balances the system more quickly due to more informed decision making. The black-box system must perform migrations sequentially and incurs an additional migration.

Fig. 6(a) plots the memory allocation of the VM over time. Both systems gradually increase the VM's memory until all unused RAM is exhausted. Since Black can only respond to swapping, it lags in responsiveness.

At t=380 seconds, Gray determines that there is insufficient RAM for the VM and migrates it to the second PM with 1GB RAM; Black initiates the same migration shortly afterward. Both continue to increase the VM's memory as the load rises. Throughout the experiment, Black writes a total of 32MB to swap, while Gray only writes 2MB. Note that a lower memory hotspot threshold in Gray can prevent swapping altogether, while Black can not eliminate swapping due to its reactive nature.

Result: A key weakness of the black-box approach is its inability to infer memory usage. Using this information, the gray-box system can reduce or eliminate swapping without significant overprovisioning of memory.

7.5 Gray v. Black: Apache Performance

Next, we compare the effectiveness of black- and gray-box approaches in provisioning resources for overloaded

VMs. Recall from Section 5 that when resources are fully utilized, they hamper the black-box approach from accurately determining the needs of overloaded VMs. Our experiment demonstrates how a black-box approach may incur extra migrations to mitigate a hotspot, whereas a gray-box approach can use application-level knowledge for faster hotspot mitigation.

Our experiment employs three physical servers and four VMs. Initially, VM_1 through VM_3 reside on PM_1, VM_4 resides on PM_2, and PM_3 is idle. We use httperf to generate requests for CPU intensive PHP scripts on all VMs. At t=80s, we rapidly increase the request rates on VM_1 and VM_2 so that actual CPU requirement for *each* virtual machine reaches 70%. The request rates for VM_3 and VM_4 remain constant, requiring 33% and 7% CPU respectively. This creates an extreme hotspot on PM_1. We configure Sandpiper with an aggressive 6 second measurement interval so that it can respond quickly to the increase in workload.

Without accurate estimates of each virtual machine's resource requirements, the black-box system falters in its decision making as indicated in Figure 7. Since the CPU on PM_1 is saturated, each virtual machine receives an equal portion of processing time and appears equivalent to Sandpiper. Sandpiper must select a VM at random, and in the worst case, tries to eliminate the hotspot by migrating VM_3 to PM_3. Since VM_1 and VM_2 continue to reside on PM_1, the hotspot persists even after the first migration. Next, the black-box approach assumes that VM_2 requires only 50% of the CPU and migrates it to PM_2. Unfortunately, this results in PM_2 becoming overloaded, so a final migration must be performed to move VM_4 to PM_3.

We repeat this scenario with the Apache gray-box module running inside of each virtual machine. Since the gray-box monitor can precisely measure the incoming request rates, Sandpiper can accurately estimate the CPU needs of VM_1 and VM_2. By using this information, Sandpiper is able to efficiently respond to the hotspot by immediately migrating VM_3 to PM_2 and VM_2 to PM_3. Figure 8 depicts the improved performance of the gray-box approach. Note that since Sandpiper requires the hotspot to persist for k out of n intervals before it acts, it is not until $t = 98$s that either system considers itself overloaded. Once a hotspot is flagged, the gray-box approach can mitigate it within 40 seconds with just two migrations, while the black-box approach requires 110 seconds and three migrations to do so. Although response time increases equally under both systems, the gray-box approach is able to reduce response times to an acceptable level 61% faster than the black-box system, producing a corresponding reduction in SLA violations.

Result: Application-level statistics enable the gray-box approach to better infer resource needs and improves the quality of migration decisions, especially in scenarios where resource demands exceed server capacity.

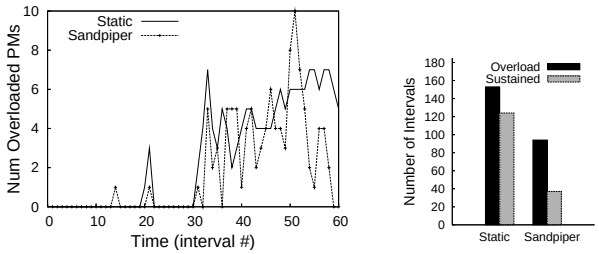


Figure 9: Sandpiper eliminates all hotspots and reduces the number of intervals experiencing sustained overload by 61% .

7.6 Prototype Data Center Evaluation

Next we conduct an experiment to demonstrate how Sandpiper performs under realistic data center conditions. We deployed a prototype data center on a cluster of 16 servers that run a total of 35 VMs. An additional node runs the control plane and one node is reserved as a scratch node for swaps. The virtual machines run a mix of data center applications ranging from Apache and streaming servers to LAMP servers running Apache, PHP, and MySQL within a single VM. We run RUBiS on our LAMP servers—RUBiS is an open-source multi-tier web application that implements an eBay-like auction web site and includes a workload generator that emulates users browsing and bidding on items.

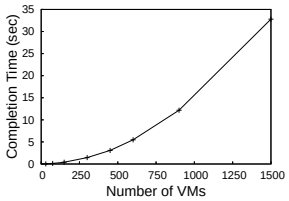
Of the 35 deployed VMs, 5 run the RUBiS application, 5 run streaming servers, 5 run Apache serving CPU-intensive PHP scripts, 2 run main memory database applications, and the remaining 15 serve a mix of PHP scripts and large HTML files. We use the provided workload generators for the RUBiS applications and use httperf to generate requests for the other servers.

To demonstrate Sandpiper's ability to handle complex hotspot scenarios, we orchestrate a workload that causes multiple network and CPU hotspots on several servers. Our workloads causes six physical servers running a total of 14 VMs to be overloaded—four servers see a CPU hotspot and two see a network hotspot. Of the remaining PMs, 4 are moderately loaded (greater than 45% utilization for at least one resource) and 6 have lighter loads of between 25 and 40% utilization. We compare Sandpiper to a statically allocated system with no migrations.

Figure 9 demonstrates that Sandpiper eliminates hotspots on all six servers by interval 60. These hotspots persist in the static system until the workload changes or a system administrator triggers manual migrations. Due to Xen's migration overhead, there are brief periods where Sandpiper causes more physical servers to be

Log Type	Bytes
Black-box	178
Gray: Mem	60
Gray: Apache	50
Total	288

(a)



(b)

Figure 10: Sandpiper overhead and scalability

overloaded than in the static case. Despite this artifact, even during periods where migrations are in progress, Sandpiper reduces the number of intervals spent in sustained overload by 61%. In all, Sandpiper performs seven migrations and two swaps to eliminate all hotspots over a period of 237 seconds after hotspot detection.

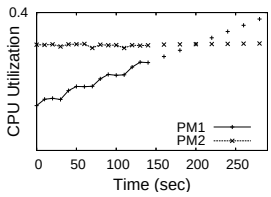
Result: Sandpiper is capable of detecting and eliminating simultaneous hotspots along multiple resource dimensions. Despite Xen's migration overhead, the number of servers experiencing overload is decreased even while migrations are in progress.

7.7 System Overhead and Scalability

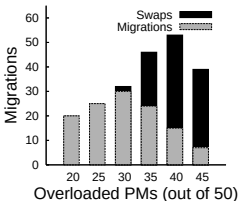
Sandpiper's CPU and network overhead is dependent on the number of PMs and VMs in the data center. With only black-box VMs, the type of application running in the VM has no effect on Sandpiper's overhead. If gray-box modules are in use, the overhead may vary depending on the size of application-level statistics gathered.

Nucleus Overheads: Sandpiper's usage reports are sent from each nucleus to the Control Plane every measurement interval (10 seconds by default). The table in Figure 10(a) gives a breakdown of overhead for each report type. Since each report uses only 288 bytes per VM, the resulting overhead on a gigabit LAN is negligible. To evaluate the CPU overhead, we compare the performance of a CPU benchmark with and without our resource monitors running. Even on a single physical server running 24 concurrent VMs, our monitoring overheads reduce the CPU benchmark by approximately one percent. This is comparable to the overheads reported by XenMon, which much of our code is based on [10]. We also compare the performance of an Apache server with and without our monitoring software and find no significant difference in response time or attainable throughput between the two scenarios.

Control Plane Scalability: The main source of computational complexity in the control plane is the computation of a new mapping of virtual machines to physical servers after detecting a hotspot. Although the problem is NP-hard, we only require an approximate solution, and our heuristics make the problem tractable for



(a)



(b)

Figure 11: (a) Using time series predictions (the dotted lines) allows Sandpiper to better select migration destinations, improving stability. (b) Higher levels of overload requires more migrations until there is no feasible solution.

reasonable system sizes. For data centers with up to 500 virtual servers, the algorithm completes in less than five seconds as shown in Figure 10(b). For very large data centers with thousands of virtual machines, the computation could be split up accross multiple nodes, or the center's servers can be broken up into pools, each controlled independently by its own control plane.

Result: Our measurement system has insignificant CPU and I/O requirements and has a negligible impact on performance. The computation time of our placement algorithm is the main limit on scalability, but the computations can be distributed if necessary.

7.8 Stability During Overloads

This section demonstrates how Sandpiper ensures stable system behavior by avoiding "thrashing" migrations. First, Sandpiper avoids migrations to physical machines with rising loads, since this can trigger additional migrations if the load rises beyond the threshold; time-series predictions are used to determine future load trends when selecting a physical server. Thus, Figure 11(a) shows that when a migration decision is required at t=140 sec, Sandpiper will prefer PM_2 over PM_1 as a target. Even though PM_2 has a *higher* current load, the 120 second prediction window indicates a rising load on PM_1.

Next, we demonstrate Sandpiper's behavior in the presence of increasing number of hotspots. We simulate a data center with fifty physical servers, each with three virtual servers. We increase the number of simultaneous hotspots from 20 to 45; the mean utilizations are set to 85% and 45% for servers with and without hotspots. Figure 11(b) depicts the mean number of migrations performed to resolve these hotspots over multiple runs. If fewer than half of the servers are overloaded, then all hotspots can typically be resolved with one migration per overloaded server. After this threshold, swaps are required and it is increasingly difficult to fully resolve overload until it becomes infeasible. With 35 overloaded servers, Sandpiper was able to eliminate

all hotspots 73% of the time (over multiple runs); with 40 overloaded servers, a complete solution was found only 3% of the time. In the extreme case, Sandpiper is still able to resolve 22 of the 45 hotspots before giving up. In all cases, Sandpiper first finds a solution before initiating migrations or swaps; when no feasible solutions are found, Sandpiper either implements a partial solution or gives up entirely rather than attempting wasteful migrations. This bounds the number of migrations which will ever be performed and explains the decrease in migrations beyond 40 overloaded servers, where there is no feasible solution.

7.9 Tuning Sandpiper

Sandpiper has several parameters which the system administrator can tune to make hotspot detection and mitigation more or less aggressive. Our experiments suggest the following rules of thumb:

Setting Thresholds: If overload thresholds are set too high, then the additional overhead during migration can cause additional SLA violations. Our experiments show that the average throughput of a CPU-intensive Apache server can drop by more than 50% during a migration. We suggest a CPU threshold of 75% to absorb the CPU overhead of migration while maximizing server utilization. We also suggest a 75% threshold for network utilization based on experiments in [6] which indicate that the network throughput of a highly loaded server can drop by about 20% during portions of a migration (due to network copying overheads).

Sustained Overload Requirement: Our experiments (not reported here) reveal that Sandpiper is not sensitive to a particular choice of the measurement interval $\mathcal{I}$ so long as it is between a few seconds and a few tens of seconds. For a measurement interval of 10s, we suggest $k = 3$ and $n = 5$ for the "k out of n" check; this corresponds to requiring the time period of about 3 migrations to exceed the resource threshold before we initiate a migration. The Δ paramter is used in the black-box system to increase resource allocations when utilization is saturated. This should be set equal to the maximum increase in resource requirements that a service is likely to see during a measurement interval and may vary based on workload; we use 10% in our experiments. Using more advanced time series forecasting techniques would allow Sandpiper to dynamically determine Δ.

8 Related Work

Our work draws upon recent advances in virtual machines and dynamic provisioning in data centers to address a question of increasing research and commercial interest: can virtual machine migration enable robust and highly responsive provisioning in data centers? The Xen migration work [6] alludes to this motivation. What is missing is a convincing validation and algorithms to effect migration, which is the focus of this paper.

The idea of process migration was first investigated in the 80's [22]. Support for migrating groups of processes across OSes was presented in [16], but applications had to be suspended and it did not address the problem of maintaining open network connections. Virtualization support for commodity operating systems in [7] led towards techniques for virtual machine migration over long time spans, suitable for WAN migration [19]. More recently, Xen [6] and VMWare [15] have implemented "live" migration of VMs that involve extremely short downtimes ranging from tens of milliseconds to a second. VM migration has been used for dynamic resource allocation in Grid environments [18, 21, 8]. A system employing automated VM migrations for scientific nano-technology workloads on federated grid environments was investigated in [18]. The Shirako system provides infrastructure for leasing resources within a federated cluster environment and was extended to use virtual machines for more flexible resource allocation in [8]. Shirako uses migrations to enable dynamic placement decisions in response to resource broker and cluster provider policies. In contrast, we focus on data center environments with stringent SLA requirements that necessitate highly responsive migration algorithms for online load balancing. VMware's Distributed Resource Scheduler [24] uses migration to perform automated load balancing in response to CPU and memory pressure. DRS uses a userspace application to monitor memory usage similar to Sandpiper's gray box monitor, but unlike Sandpiper, it cannot utilize application logs to respond directly to potential SLA violations or to improve placement decisions.

Dedicated hosting is a category of dynamic provisioning in which each physical machine runs at most one application and workload increases are handled by spawning a new replica of the application on idle servers. Physical server granularity provisioning has been investigated in [1, 17]. Techniques for modeling and provisioning multi-tier Web services by allocating physical machines to each tier are presented in [23]. Although dedicated hosting provides complete isolation, the cost is reduced responsiveness - without virtualization, moving from one physical machine to another takes on the order of several minutes [23] making it unsuitable for handling flash crowds. Our current implementation does not replicate virtual machines, implicitly assuming that PMs are sufficiently provisioned.

Shared hosting is the second variety of dynamic provisioning, and allows a single physical machine to be shared across multiple services. Various economic and

resource models to allocate shared resources have been presented in [5]. Mechanisms to partition and share resources across services include [2, 5]. A dynamic provisioning algorithm to allocate CPU shares to VMs on a single physical machine (as opposed to a cluster) was presented and evaluated through simulations in [14]. In comparison to the above systems, our work assumes a shared hosting platform and uses VMs to partition CPU, memory, and network resources, but additionally leverages VM migration to meet SLA objectives.

Estimating the resources needed to meet an application's SLA requires a model that inspects the request arrival rates for the application and infers its CPU, memory, and network bandwidth needs. Developing such models is not the focus of this work and has been addressed by several previous efforts such as [12, 1].

9 Conclusions And Future Work

This paper argued that virtualization provides significant benefits in data centers by enabling virtual machine migration to eliminate hotspots. We presented Sandpiper, a system that automates the task of monitoring and detecting hotspots, determining a new mapping of physical to virtual resources and initiating the necessary migrations in a virtualized data center. We discussed a black-box strategy that is fully OS- and application-agnostic as well as a gray-box approach that can exploit OS- and application-level statistics. An evaluation of our Xen-based prototype showed that VM migration is a viable technique for rapid hotspot elimination in data center environments. Using solely black-box methods, Sandpiper is capable of eliminating simultaneous hotspots involving multiple resources. We found that utilizing gray-box information can improve the responsiveness of our system, particularly by allowing for proactive memory allocations and better inferences about resource requirements. In future work, we plan to extend Sandpiper to support replicated services by automatically determining whether to migrate a VM or to spawn a replica in order to garner more resources.

Acknowledgements: We would like to thank our anonymous reviewers and our shepherd, Anees Shaikh, for their helpful comments. This research was supported by NSF grants EEC-0313747, CNS-0323597, CNS-0325868 and a gift from Intel.

References

[1] K. Appleby, S. Fakhouri, L. Fong, M. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano - sla-based management of a computing utility. In *Proc. IFIP/IEEE Symposium on Integrated Management*, May 2001.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proc. ACM SIGMETRICS '00*.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. SOSP'03*, pages 164–177, October 2003.

[4] G. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis Forecasting and Control Third Edition*. Prentice Hall, 1994.

[5] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proc. SOSP '01*.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. NSDI '05*, May 2005.

[7] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular disco: Resource management using virtual clusters on shared-memory multiprocessors. In *Proc. SOSP'99*, pages 154–169, December 1999.

[8] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase. Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration. In *Proc. VTDC '06*.

[9] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. In *Proc. Middleware '06*, October, 2006.

[10] D. Gupta, R. Gardner, and L. Cherkasova. Xenmon: Qos monitoring and performance profiling tool. Technical Report HPL-2005-187, HP Labs, 2005.

[11] S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Geiger: Monitoring the buffer cache in a virtual machine environment. In *Proc. ASPLOS'06*, pages 13–23, October 2006.

[12] A. Kamra, V. Misra, and E. Nahum. Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites. In *Proc. IWQoS '04*, June 2004.

[13] L. Kleinrock. *Queueing Systems, Volume 2: Computer Applications*. John Wiley and Sons, Inc., 1976.

[14] D. Menasce and M. Bennani. Autonomic Virtualized Environments. In *IEEE ICAS 06*.

[15] M. Nelson, B. Lim, and G. Hutchins. Fast Transparent Migration for Virtual Machines. In *Proc. USENIX* 2005.

[16] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: A system for migrating computing environments, In *Proc. OSDI* 2002.

[17] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *Proc. IWQoS 2002*.

[18] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In *Proc. IEEE ICAC '06*.

[19] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proc. OSDI '02*.

[20] V. Sundaram, T. Wood, and P. Shenoy. Efficient Data Migration in Self-managing Storage Systems. In *Proc. ICAC '06*.

[21] A. Sundararaj, A. Gupta, and P. Dinda. Increasing Application Performance in Virtual Environments through Run-time Inference and Adaptation. In *Proc. HPDC '05*.

[22] M. M. Theimer, K. A. L., and D. R. Cheriton. Preemptable Remote Execution Facilities for the V-System. In *Proc. SOSP* December 1985.

[23] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning for multi-tier internet applications. In *Proc. ICAC '05*, June 2005.

[24] VMware Dynamic Resource Scheduler. http://www.vmware.com/products/vi/vc/drs.html.

View publication stats

The Journal of Supercomputing
https://doi.org/10.1007/s11227-021-04107-6

Deep learning-based multivariate resource utilization prediction for hotspots and coldspots mitigation in green cloud data centers

Yashwant Singh Patel[1] · Rishabh Jaiswal[2] · Rajiv Misra[3]

Accepted: 20 September 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

Dynamic virtual machine (VM) consolidation is a constructive technique to enhance resource usage and is extensively employed to minimize data centers' energy consumption. However, in the current approaches, consolidation techniques are heavily relied on reducing the actively used physical servers (PMs) based on their current resource utilization without considering future resource demands. Also, many of the reported works for cloud workload prediction applied univariate time series-based forecasting models and neglected the dependency of other resource utilization metrics. Thus, resulting in inaccurate predictions, unnecessary migrations, high migration costs, and increased service level agreement violations (SLAVs) may nullify the consolidation benefits. To efficiently address this issue, we propose a multivariate resource usage prediction-based hotspots and coldspots mitigation approach that considers both the current and future usage of resources with $O(sk)$ time complexity, where s and k denote the number of PMs and VMs, respectively. The proposed technique uses a clustering-based stacked bidirectional (Long Short-Term Memory) LSTM deep learning network to predict the future memory and CPU usage of PMs and VMs with high accuracy and $O((Q(Q+W)*\Theta)$ computational complexity, where Q, W, and Θ represent the number of hidden layer cells, outputs, and training epochs, respectively. Through extensive simulations based on Google's cluster workload traces, we demonstrate that our proposed method obtains substantial improvements in terms of prediction performance, energy-efficiency, actively used PMs, VM migrations, and SLA violations over the benchmark approaches.

Keywords Green cloud computing · Deep learning · Dynamic VM consolidation · Energy-efficiency · SLA · Migration · Resources' utilization · Prediction approaches · Performance evaluation

✉ Yashwant Singh Patel
yashwant.patel@thapar.edu

Extended author information available on the last page of the article

Published online: 07 October 2021

1 Introduction

Cloud computing is rapidly emerging and evolving as a successful computing paradigm for providing on-demand and dynamic provisioning of IT infrastructure, services, and resources in a pay-per-use manner. The cloud provides not only readily available infrastructure but also the ability of rapid auto-scaling to support the massive and fluctuating big data workloads. In the cloud, the data centers are connected via networks and offer flexible and cost-effective services for individuals, organizations, and enterprises. To satisfy the diverse demands from different users, the commercial cloud Infrastructure-as-a-Service or IaaS providers, such as Amazon EC2, IBM, and Microsoft Azure, offer different VM instances with heterogeneous resources (or dimensions) (e.g., memory, CPU, storage and network bandwidth) based on pay-per-use business model [1]. In this regard, the virtualization technology such as VMware [2] and Xen [3] plays a vital role for efficient resource management in a multi-user environment. With a hypervisor or virtual machine monitor (VMM), it permits several VMs to share a single PM's resources. By adopting such virtualized infrastructure technology, several public cloud service providers (CSPs) such as Microsoft, Amazon, Yahoo, and Google build massive data centers geographically distributed all over the globe to offer cloud services with low cost, global coverage, low latency, and high application availability.

However, cloud infrastructures' rising demands are drastically increasing the global data center energy consumption [4]. As per the recent reports on world's servers power consumption, the electricity consumption of servers worldwide accounts for 3% of the global electricity production [5]. Such high data center energy consumption not only leads to considerable operation expenses, but also generates substantial carbon dioxide (CO_2) emissions. Gartner estimated that the information and communication technology (ICT) industry ecosystem accounts for greater than 2% of global CO_2 emissions, which is equal to the aviation industry's emissions from fuel [6, 7]. In the context of ICT's future energy demand, analysts also forecasted that the data center electricity usage would increase approximately 15 times by 2030, which may result in around 8% of the projected global demand of electricity [8]. Therefore, even though several phenomenal developments on cloud infrastructures are made, designing effective energy-aware resource provisioning strategies to enhance energy-efficiency of data centers have become essential, as depicted in Fig. 1. Additionally, supporting a good quality of service (QoS) level is necessary for CSPs to satisfy users' expectations concerning performance. Service level agreement (SLA) is the concrete implementation to define QoS requirements, which are contracts to describe service level details offered to consumers, such as downtime ratio, system throughput, and response time. Therefore, minimizing the overall energy consumption of data centers, while restricting the SLA violations (SLAVs), is the principle objective of this study.

The research on cloud data centers shows that the physical machines run at 10 to 50% of their maximum CPU usage [9]. Additionally, the majority of PMs are

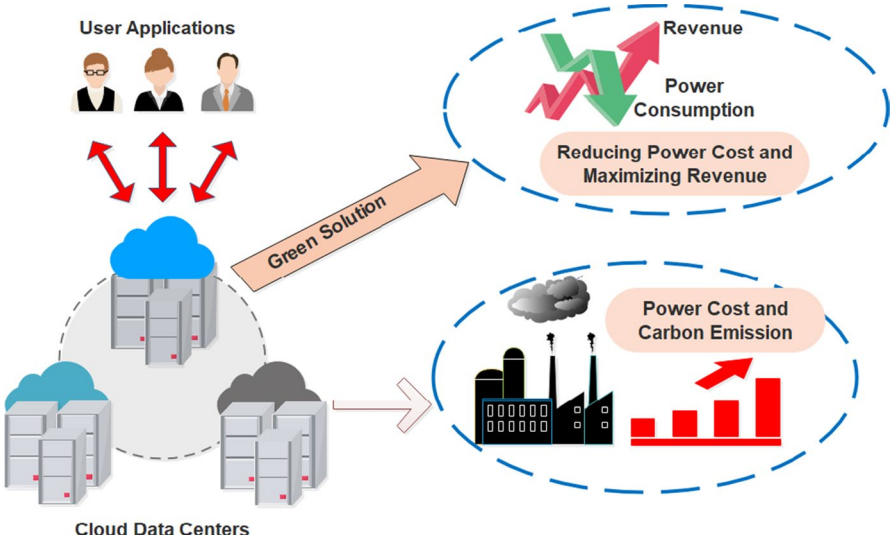


Fig. 1 Green Cloud Computing

idle PMs, which consume about 70% of their peak power [10]. To reduce energy waste and improve resource utilization, the CSPs apply dynamic VM consolidation approaches. By leveraging hardware virtualization technique [3], the VM instances are dynamically consolidated and packed in fewer PMs while considering their current resource demands. In addition, by using live migration technology [11], the virtualization enables the CSPs to dynamically allocate VMs in least active PMs and allows switching the idle PMs into sleep mode. Migration is transparent and beneficial when a physical server is highly overloaded (i.e., creating hotspot) or underloaded (i.e., creating coldspot) [12, 13]. However, consolidation policies reduce energy consumption significantly but live VM migration results in increased SLAVs. Consequently, an effective VM consolidation approach is crucial for reducing the energy expenses while meeting SLAs. In a cloud data center comprising thousands of heterogeneous PMs, the consolidation process comprises three phases [12]:

1. *Hotspots and coldspots detection* Determining the PM is become overloaded (i.e., creating hotspots) or underloaded (i.e., creating coldspots);
2. *VM selection* What VMs can be chosen for migration from the hotspots and coldspots while achieving the objectives such as reducing migration cost or total number of migrations;
3. *Destination PM selection* What are the target PMs for VMs that should be migrated to mitigate the hotspots and coldspots.

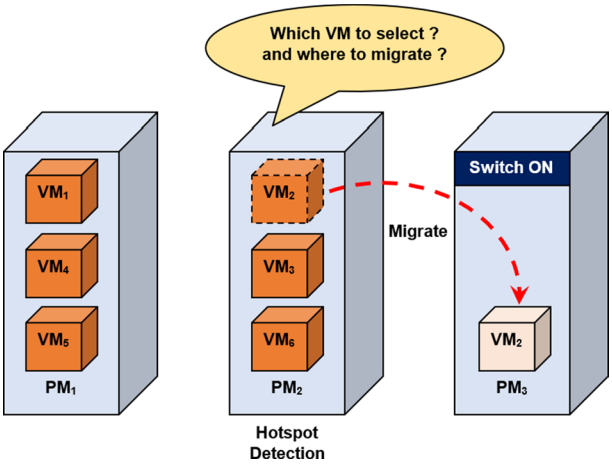


Fig. 2 Hotspot mitigation

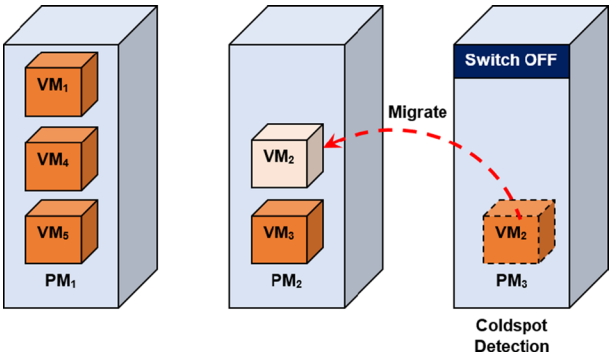


Fig. 3 Coldspot Mitigation

1.1 Example

Consider the examples represented in Figs. 2 and 3 to illustrate the basic operations of dynamic VM consolidation, including hotspot mitigation and coldspot mitigation, respectively. Figure 2 demonstrates a hotspot mitigation example with three PMs $< PM_1, PM_2, PM_3 >$ and six VMs $< VM_1, \ldots, VM_6 >$ are placed on them. Assume the hotspot is detected on PM_2 due to resource over-utilization. In this case, a few of the VM instances should be migrated to some other suitable PMs. Let's say, VM_2 is selected and migrated from PM_2 to PM_3.

In Fig. 3, we have presented an example of coldspot mitigation, where five VMs, i.e.,VM_1, VM_2, VM_3, VM_4, and VM_5 are hosted on three PMs. The coldspot mitigation process identifies the underloaded machines and tries to migrate all VMs from that PM to any other suitable PM without creating any other hotspots. In the

given example, the consolidation process determines PM_3 as a least loaded PM and migrate VM_2 from PM_3 to PM_2. With the intent that the PM_3 could be switched to power-saving mode to maximize the energy-efficiency.

1.2 Motivation

In the context of the VM consolidation, many of the reported works have addressed the VM allocation and migration problem as a multi-dimensional bin-packing problem, known to be an NP-hard problem [14]. Similar to bin-packing, the physical servers would be considered as bins and VMs mapped with items. To apply in the VM placement and migration scenario, the classical bin-packing heuristics have to be modified in three aspects: first, unlike the bin-packing, the PMs are treated as bins with different sizes. Second, the PMs and VMs are characterized by varying dimensions of resource so that optimal balancing can be performed with respect to resource usage across several dimensions. Third, the objective of classical bin-packing algorithms is only to reduce the total number of bins. Still, in the case of a consolidation, there can be multi-objective such as minimizing SLA violations, actively used PMs and migrations. Thus, to find near-optimal solutions, it requires optimization algorithms. Also, for effective energy-efficient provisioning of resources in green data centers, several schemes have been provided. However, from the reported studies, we have found a few aspects of improvements:

1. Many of the existing solutions only consider the current resource utilization for hotspots and coldspots mitigation.
2. Reported works on resource utilization prediction in the cloud only consider the univariate time series-based classical methods. Such methods apply only the past trends of the target resource usage metric, i.e., CPU utilization, only to generate future predictions.
3. Assignment of VMs to PMs to reduce energy cost as much as possible without considering the SLA violations.
4. Live VM migration is necessary as the VMs arrive and depart periodically in highly dynamic data centers, but it results in high migration cost (e.g., network cost).
5. Migration decision to mitigate the hotspots and coldspots, when there are not sufficient resources on least loaded PMs.

1.3 Contributions

In this paper, our approach is to predict future utilization of resources on the basis of past workload traces of PMs and VMs. Current and predicted future utilization metrics are applied for efficient hotspots and coldspots mitigation to improve the data centers' energy-efficiency while preserving the QoS guarantee. The technical contributions of this work are highlighted as follows:

1. Demonstrate the advantages of using different feature selection methods to determine relevant features subset to predict future resource utilization.
2. Design of multivariate resource utilization prediction approach based on the clustering-based stacked bidirectional LSTM deep learning model to forecast future resource usage as per the past trace workload on the considered hosts.
3. Effective dynamic VM consolidation algorithms using current and future resource utilization is presented, namely prediction-based hotspots detection, prediction-based coldspots detection, energy and QoS based VM selection, and destination PMs selection for migrated VMs.
4. Through real workload traces, we investigate the relationship in respect of prediction accuracy, migration cost, energy consumption, actively used PMs, and SLA violations with resource utilization thresholds.

1.4 Article organization

The remaining of the current paper is structured as follows: Sect. 2 provides the related studies on the dynamic VM consolidation and prediction approaches for cloud data centers. The problem definition is presented in Sect. 3. The system model and performance models are provided in Sect. 4. In Sect. 5, we present different feature selection techniques, proposed clustering-based stacked bidirectional LSTM prediction model, the VM consolidation algorithms for hotspots and coldspots mitigation, including hotspots detection, coldspots detection, VM selection, and destination PMs selection, and complexity analysis. Simulation experiments with real-world traces are demonstrated in Sect. 6. At last, Sect. 7 concludes the work with some of the future directions.

2 Related works

We study the problem of VM consolidation in data centers. Several studies comprising VM consolidation and resource prediction in the cloud are discussed to address this problem.

2.1 VM consolidation

In recent years, there have been many methodologies reported in the literature on VM consolidation techniques under different settings [4, 15, 16]. In [13] Timothy wood et al. presented the sandpiper system that combines the product of three dimensions, i.e., CPU, network, and memory utilization, into a single dimension volume metric. Sandpiper has implemented gray-box and black-box approach (BG) to automate the cloud system for continuous monitoring of hotspots. To capture the degree of overload, it uses the volume metric to sort all the overloaded servers and apply the volume-to-size ratio (VSR) for VMs in each server. The BG strategy then selects the highest volume server first and the highest VSR VM for migration. To allocate these VMs, it sorts the destination PMs based on their increasing volume

metric. The major limitation of sandpiper is that it does not provide good performance while characterizing underloaded and overloaded servers. It is also limited to homogeneous servers and does not enable the coldspot mitigation.

Beloglazov et al. [12] proposed thresholds (i.e., lower and upper) based on adaptive heuristics. For host overloading detection, they presented MAD: median absolute deviation, IQR: interquartile range, LR: local regression, and robust LR-based approaches that can be applied via the statistical investigation of the historical resource usage data. They also designed three policies for selecting VMs: maximum correlation, random selection, and MMT: minimum migration time. Furthermore, to identify the destination host, they have implemented a Power-aware Best Fit Decreasing (PBFD) technique, combining Best Fit Decreasing (BFD) bin-packing scheme and a power-aware model. They implemented these policies on CloudSim [17] with PlanetLab workload traces. Using experimental results, they showed that the LR-based method along with the MMT policy of selecting VMs performs better than existing approaches in regard to the number of migrations, energy consumption, SLAVs, and ESV metric (where ESV = SLAV × energy consumption).

Farahnakian et al. [18] provided LR-based method to predict CPU utilization of PMs and simulated it using CloudSim. In [19], researchers have applied the K-nearest neighbor regression approach to demonstrate that it could further enhance the SLAV and energy consumption more than the linear regression method. In [20], the authors have presented a metaheuristic algorithm named Ant Colony System (ACS) for VM consolidation. Through experimental results, they showed that it could improve the overall system performance in terms of VM migrations, energy consumption, and QoS requirements. F. Farahnakian et al. [21] designed a utilization prediction-based dynamic VM consolidation method. It applies a regression-based strategy to predict future usage of resources. Their approach uses both the current and future usage of resources to estimate the next time step memory and CPU usage of PMs and VMs. Through simulation results, they demonstrated that the consolidation process could improve the performance of SLA violations, migrations, and energy consumption.

Moghaddam et al. [22] proposed machine learning (ML)-based resource prediction models. They have tuned a forecasting framework for each VM independently, which uses the VM's past behavior for predicting their CPU utilization. Haghshenas et al. [23] designed a regression-based technique to predict the resource utilization of PMs and VMs. They have applied a linear regression-based algorithm for destination host selection during the migration of VMs. Furthermore, they proposed an improved version of the approach to predict the host usage for underutilized host selection. Hsieh et al. [24] suggested a VM consolidation technique for the host overload and underload detection. They have used a Gray-Markov-based model to estimate future resource utilization. Tarafdar et al. [25] applied a QoS and energy-aware consolidation technique. They have applied a Markov chain-based model to detect the underloaded and overloaded PMs. Additionally, they applied a VM selection and placement scheme using the linear weighted sum technique.

Chen et al. [26] proposed an Exponentially Weighted Moving Average or EWMA scheme to identify host overloading. They further evaluated their approach using local regression and identified that when the constant value is 0.025, the EWMA

performs better with respect to SLAVs, energy-efficiency, and ESV. Donnell et al. [27] provided Gossip Contracts (GC)-based dynamic virtual machine consolidation approach for a decentralized solution. Their approach performs well in respect of power consumption and SLA Violations. Naeen [28] applied Markov chain models for VM consolidation with QoS constraints.

Li et al. [29] designed a quality and energy-aware VM consolidation (EQ-VMC) strategy to improve the service quality and energy-efficiency. They have developed a discrete differential evolution heuristic to explore the global optimum solution for VM assignment. El-Moursy et al. [30] designed a multi-dimension-based regression algorithms to identify overloaded PMs, which integrate memory, CPU, network BW usage. Ranjbari et al. [31] suggested a technique of learning automata to improve resource usage while reducing energy consumption.

By reviewing the previous works, we have observed that most of the current solutions apply a single prediction model for resource utilization prediction for all the VMs or PMs. In addition, most of the studies have meticulously focused on CPU utilization and neglected the computational complexity and optimization of trade-offs between SLA violation, migrations and energy consumption. Most of the reviewed studies have examined future resource utilization of PMs and migration time independently. While some of the current solutions only examine the current utilization of resources and address the single objective of energy cost minimization and may not be feasible to work in any arbitrary stage of the cloud data centers such as detecting and mitigating multiple hotspots and coldspots at any time instant. Therefore, in the next subsection, we explore the prominent prediction models applied for cloud resource usage prediction.

2.2 Prediction models for cloud workloads

In the cloud, the problem of resource utilization prediction can be framed as a time series forecasting model, where we can analyze the past data of cloud resource utilization to estimate future values. Several statistical models, such as Holt-winter [32], ARIMA (Autoregressive integrated moving average) [33], seasonal ARIMA (SARIMA) [34], Feed-forward neural network [35], and Markov models [36, 37], are proposed for utilization prediction. In contrast, Zhang et al. [33] applied the ARIMA method to estimate the future cloud resource utilization. This model considers the last t observations of past data RH_t to predict the resource usage RH_{t+1} at time $t+1$. It can be expressed as:

$$\mathrm{RH}_{t+1} = \rho_0 \mathrm{RH}_t + .. + \rho_{m-1} \mathrm{RH}_{t-n+1} + \epsilon_{t+1} + \lambda_0 \epsilon_t + \cdots + \lambda_{k-1} \epsilon_{t+1-k}, \tag{1}$$

where the notation $\rho + i$ and λ_j represent the constants. ϵ_t is the independent error term. The other parameters n and k show the number of lags or previously measured resource usage values and error terms correspondingly. ARIMA is one of the well-known time series prediction models. However, it is failed to detect the presence of nonlinear patterns present in the time series data. Caglar et al. [35] presented an iOverbook model, which applies a two-layer feed-forward artificial neural network

(ANN). It also generalizes the nonlinear and linear correlation between the input and output using a single hidden layer. It can be defined as:

$$\Delta\hat{z}^t = \sum_{q=1}^{n} \alpha_q \Upsilon\left(\sum_{p=1}^{m} \theta_{pq} \Delta z^{t-p}\right) \tag{2}$$

where $\Delta\hat{z}^t$ defines the estimated resource utilization value at time t. The term m denotes total past lags, and n shows the total neurons present in the hidden layer of network. The training weight parameters are denoted by α and θ. The term $\Upsilon(.)$ shows the activation function. However, most of the abovementioned studies have presumed that observations of resource usage in a long-time period are unassociated with each other. In addition, these techniques have also presumed that the time series data is memoryless and stationary. But Ghorbani et al. [38] identified the existence of long-range dependence (LRD) in the Google's cluster workload trace [39]. The LRD may occur during the time series analysis, where the past-time lags may affect the next step value. The LRD may exist if the dependence decays more gradually than any decaying exponential and mostly similar to a power-like decay. Intuitively, the LSTM (Long short-term memory) models are more suitable to handle this issue, as it can model the LRD in time series.

Song et al. [40] adopted univariate LSTM networks, where they have analyzed the past CPU resource utilization to forecast the future CPU usage trends. LSTMs are a robust kind of recurrent neural networks (RNNs) and work well on sequence-based tasks having long-term dependencies. The basic difference between the component of RNN architecture and standard LSTM architecture is the hidden layer. The LSTM's hidden layer is also termed as LSTM cell. The basic LSTM block is shown in Fig. 4. It comprises a memory cell C_t, an input gate i_t, an output gate o_t, and also a forget gate f_t. The weight matrices are W_{xc}, W_{hc}, W_{xi}, W_{hi}, W_{xo}, W_{ho}, W_{xf}, and W_{hf}. The activation functions are Sigmoid and *tanh* denoted by σ and *tanh*, respectively.

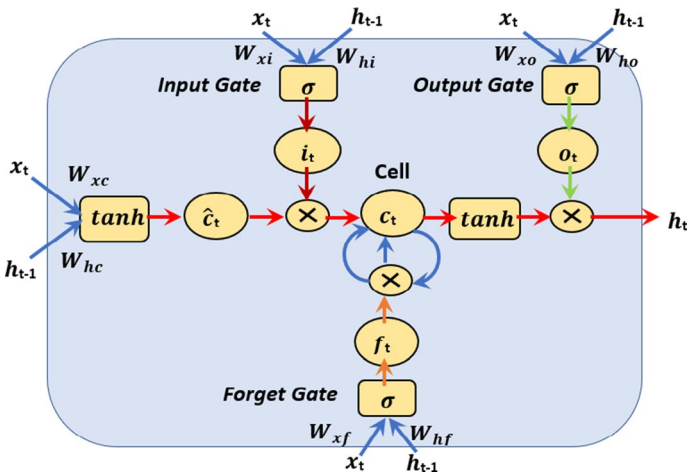


Fig. 4 LSTM memory block

Moreover, the x denotes the point-wise multiplication. At time instant t, x_t indicates the input and h_t shows the hidden state. The memory cell's candidate state is denoted by $\hat{C}_t$. It decides how much input information is received by the cell state. The following expresses the formulation for each gate including input candidate, hidden states, and cell state:

$$f_t = \sigma(x_t * W_{xf} + b_f + h_{t-1} * W_{hf}) \tag{3}$$

$$C_t = C_{t-1} * f_t + \hat{C}_t * i_t \tag{4}$$

$$\hat{C}_t = tanh(b_c + x_t * W_{xc} + h_{t-1} * W_{hc}) \tag{5}$$

$$i_t = \sigma(b_i + x_t * W_{xi} + h_{t-1} * W_{hi}) \tag{6}$$

$$o_t = \sigma(b_o + x_t * W_{xo} + h_{t-1} * W_{ho}) \tag{7}$$

$$h_t = o_t * tanh(C_t) \tag{8}$$

It is observed that most of the workload prediction networks only analyze the past resource utilization trends of the required resource usage metric to predict subsequent resource usage. But the other resource usage metrics are inter-related to each other, and it may impact the overall performance of desired resource metrics. Also, in [41, 42], it is shown that the multivariate approaches deliver high performance and promising results than the univariate approaches. Hence, it is inevitable to analyze the multivariate time series prediction to obtain correct behavior and correlation between the resource usage metrics. Several works have used feature selection methods to recognize the set of relevant features. Such as Dannecke et al. [43] applied the correlation-based feature selection method, where they have used the Pearson correlation technique to distinguish the relevant features set for energy demand prediction. Sun et al. [44] proposed a Granger causality-based model for multivariate time series investigation of Parkinson's telemonitoring and monitoring of water quality. In this work, we have used different features such as CPU usage, memory assigned, utilized memory, total page cache usage, unmapped page cache, utilization of maximum memory observed, disk I/O time, disk capacity space utilization, usage of maximum CPU (over an interval), maximum disk I/O time (over an interval), memory accesses per instruction, and cycles Per Instruction (CPI). By analyzing the diverse set of features, we can identify the correlation between different features.

Most of the studies in the cloud are limited to the unidirectional LSTM (i.e., forward dependencies) to predict future events, which only preserves the information of past resource usage. In [45], Zhao et al. proposed a bidirectional time series model that can acquire the long-range context in both dependencies (i.e., forward and backward) using the extreme learning machines approach. They have observed

that the prediction accuracy is highly improved in the bidirectional approach. For the forecasting of laser data, Wakuya et al. [46] applied the bi-directional time series computation and concluded that the prediction model's quality is much better than the unidirectional model. Cui et al. [47] proposed a deep stacked bidirectional and unidirectional LSTM (SBU-LSTM) neural network, which applies both forward and backward dependencies of time series data, to predict the network wide traffic speed. Furthermore, Gupta et al. [48, 49] studied the multivariate LSTM and BLSTM models for cloud workload prediction using Google cluster trace. However, the application of stacked bidirectional LSTM (BiLSTM) deep learners has not been reported for the interpretation of cloud workload patterns. To the best of authors' knowledge, this study is a first attempt to use a clustering-based stacked BiLSTM based multivariate resource utilization prediction model for hotspots and coldspots mitigation in the cloud. Tables 1 and 2 highlight the key differences of existing techniques against our proposed approach.

3 Problem definition

To achieve maximum expected benefit in a virtualized data center, an efficacious consolidation strategy optimizes the VMs assignment for reducing the actively used PMs. This gain solely depends on two primary factors: minimize the SLA violations and migrations of VMs. We can realize these advantages in advance by placing the VMs to PMs on the basis of their future resource usages. Thus, to understand the limitation of the classical VM consolidation approach without predicting next time step resource utilization, let's consider two examples represented in Figs. 5 and 6, respectively. Assume that there are two PMs and three VMs are placed on PMs. Through Fig. 5a, we can observe the CPU utilization of PMs namely PM_1 and PM_2 as 55% and 40% separately. As PM_1 has sufficient resources to place VM_3, a classical consolidation approach will migrate VM_3 from PM_2 to PM_1 at time instant t and switch PM_2 to the low-power state or sleep state. At the time instant $t + 1$ in Fig. 5b, when the CPU utilization demand of VM_1 is increased from 30 to 38%. The PM_1 is become overloaded due to the lack of adequate resources, and some SLA violations occur. Thus, in Fig. 5c, VM_3 is migrated from PM_1 to PM_2 to eliminate the SLA violations further. In this scenario, a robust consolidation approach can avoid useless migrations and minimize SLA violations' rate if it could predict the future resource demand of any VM prior to migration.

Another scenario is depicted in Fig. 6. In this case, at time instant t in Fig. 6a, VM_3 is migrated from PM_2 to PM_1 since it has ample resource to fulfill the demand of VM_3. In Fig. 6b, PM_2 is switched to the low-power state or sleep mode. We noticed that the CPU usage of VM_3 is further increased from 40 to 50%, and a hotspot is created. To mitigate the hotspot and avoid the SLA violations further, VM_3 is migrated to PM_2 at time instant $t + 2$ as represented in Fig. 6c.

Through both the illustrated examples, we observe that the unnecessary migrations and potential SLA violations could be eliminated if a consolidation process considers the future resource usage of PMs and VMs during the assignment of VMs. In this article, the problem of dynamic VM consolidation aims towards robust

Table 1 Comprehensive review of existing VM consolidation approaches

Year & Work	Utilized technique	Performance metrics				Type of prediction		Deep learning	Type of mitigation		Workloads
		SLA	Energy	Migration	Prediction error	Univariate	Multivariate		Hotspots	Coldspots	
2020 [22]	Machine learning, best fit decreasing	✓	✓	✓	✓	✓	×	×	✓	✓	PlanetLab
2020 [23]	Regression-based	✓	✓	×	✓	✓	×	×	✓	✓	PlanetLab
2020 [24]	Gray-Markov-based	✓	✓	✓	×	✓	×	×	✓	✓	PlanetLab
2020 [25]	Markov chain-based	✓	✓	✓	×	✓	×	×	✓	✓	PlanetLab
2020 [27]	Gossip and contract net protocols	✓	✓	✓	×	×	×	×	✓	✓	Gossip and Contract Net protocols
2020 [28]	Markov chain models	✓	✓	✓	✓	✓	×	×	✓	✓	PlanetLab
2020 [29]	Discrete differential evolution algorithm	✓	✓	✓	×	×	×	×	✓	✓	Bitbrains trace, Google-ClusterTrace, Alibaba cluster data
2019 [30]	Regression-based	✓	✓	✓	×	✓	×	×	✓	×	PlanetLab, Gaia cluster data

Table 2 (Continued.) Comprehensive review of existing VM consolidation approaches

Work	Utilized Technique	Performance metrics				Type of prediction		Deep Learning	Type of mitigation		Workloads
		SLA	Energy	Migration	Prediction Error	Univariate	Multivariate		Hotspots	Coldspots	
2019 [21]	Regression-based	✓	✓	✓	✓	✓	×	×	✓	✓	PlanetLab, Google cluster
2018 [31]	Learning Automata	✓	✓	✓	×	✓	×	×	✓	✓	PlanetLab
2018 [26]	Exponentially weighted moving average	✓	✓	✓	×	✓	×	×	✓	×	PlanetLab
2015 [20]	Ant colony optimization	✓	✓	✓	×	✓	×	×	✓	✓	PlanetLab
2013 [18]	Linear regression	✓	✓	×	×	✓	×	×	✓	✓	PlanetLab
2013 [19]	k-nearest neighbor regression	✓	✓	×	×	✓	×	×	✓	✓	PlanetLab
2012 [12]	Best fit decreasing, regression	✓	✓	✓	×	✓	×	×	✓	✓	PlanetLab
2007 [13]	Black-box & Gray-box	✓	×	✓	×	✓	×	×	✓	×	SPECjbb 2005
Proposed	Stacked BiLSTM	✓	✓	✓	✓	✓	✓	✓	✓	✓	Google cluster-usage traces

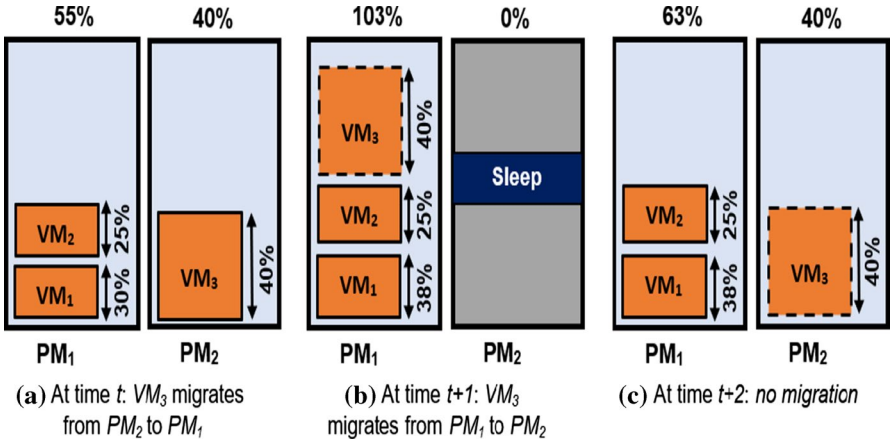


Fig. 5 Mitigation example 1

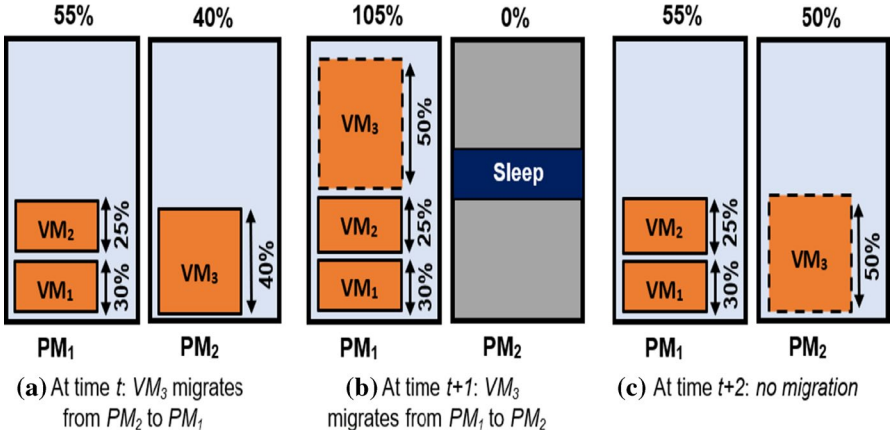


Fig. 6 Mitigation example 2

decision-making during the detection of hotspots (when the PMs are overloaded) and coldspots (when the PMs are underloaded). Once the detection of hotspots and coldspots is performed, all potential VMs are migrated from such PMs to realize the goals of maximum QoS level and reducing energy consumption, respectively.

4 System model

We consider an IaaS cloud data center environment, where its users initiate VM requests as represented in Fig. 7. The data center contains a pool of m heterogeneous PMs denoted by $L_{PM} = \{\mathrm{pm}_1, \mathrm{pm}_2, \mathrm{pm}_3, \ldots, \mathrm{pm}_m\}$. Each PM owns d types of resources namely memory, CPU, network bandwidth, and disk storage, etc., for

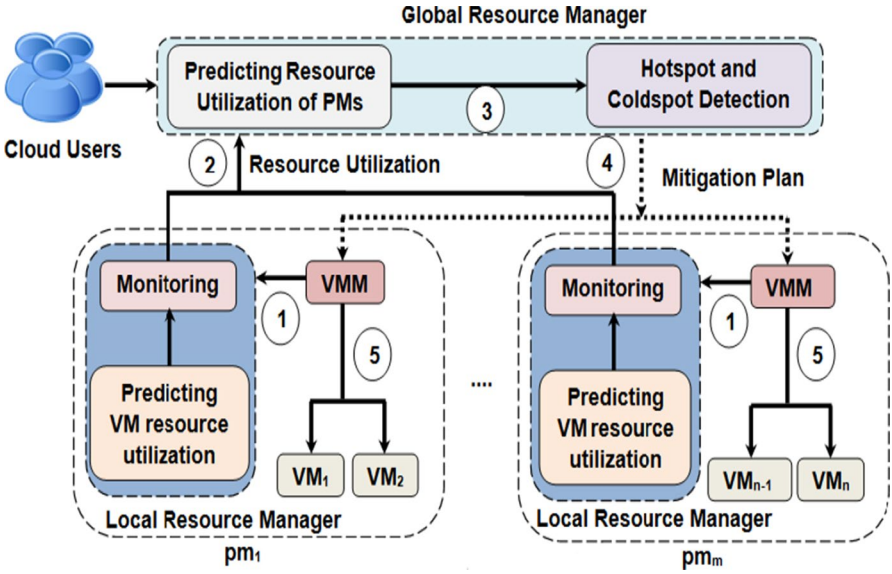


Fig. 7 The system model

which R_j^d depicts the availability of resource d in pm_j. The data center possesses ample capacity to serve all arriving and currently running requests. Next we assume a set of VMs denoted by $L_{\mathrm{VM}} = \{\mathrm{vm}_1, \mathrm{vm}_2, \mathrm{vm}_3, \ldots, vm_n\}$. Here, n indicates the number of VMs and r_i^d represents the resource demand d of vm_i. The VMs require to be allocated to PMs via a hypervisor or Virtual Machine Monitor (VMM). For initial resource assignment of VMs to PMs, the BFD algorithm is applied as it is one of the most prominent heuristic to solve the bin-packing problem. BFD algorithm always prefers a PM in such a way that the resources requested by the VM is closest to the number of available resources so that the resource wastage in the destination PM can be minimized [20]. However, due to dynamic workloads in the cloud, the requested usage of PMs and VMs fluctuate over time. Therefore, we need to optimize the initial VM placement using a consolidation process that could be applied periodically based on current and future workloads. In our proposed approach, the hotspots and coldspots mitigation algorithm is applied every 5 min in a cloud data center so that the actively used PMs and total energy consumption can be reduced based on the workload [20].

The proposed model comprises of two resource managers: (i) The Local Resource Manager (LRM) which resides on each PM as a component of the Hypervisor and (ii) The Global Resource Manager (GRM) which helps in acquiring data center view on a higher (global) level and in managing the VM requests in a data center [12]. The following steps illustrate the sequence of processes performed using these resource managers:

1. LRMs continuously monitors the current resource usage of its assigned PM and the state of its constituent (running) VMs. It then forecasts the future resource

usages of all VMs placed in a PM on the basis of historical workload traces via the clustering-based stacked bidirectional LSTM method and periodically sends the information obtained to the GRM.
2. The responsibility of the GRM is to communicate with LRMs and collect information such as current and future resource usage of all VMs to acquire the global data center view.
3. Based on the information received, the GRM identifies hotspots and coldspots. It then builds a mitigation plan and issues the necessary commands to VMMs for its execution. The commands indicate the list of VMs for migration and their corresponding target PMs on the basis of mitigation algorithms.
4. After receiving the commands from GRM, the VMMs execute the actual migration of VMs.

Next, we discuss the power and energy modeling for PMs and the migration overheads of VMs to be migrated. Also, we describe the performance metrics to measure the QoS and SLA violations.

4.1 Power and energy modeling

In data centers, the energy consumption of computing nodes is mostly analyzed by the memory, CPU, power supplies, cooling systems, and disk storage. As reported in the literature [50], the CPU consumes maximum amount of energy, when compared to the other system resources of server power consumption. The SpecPower [51] benchmark defines the relationship between the server's utilization and power consumption. Based on this benchmark, the total power consumption of a cloud server increases linearly with the increase in its CPU usage [12]. Moreover, the server's power consumption is directly proportional to its CPU usage as written in Eq. (9).

$$\mathrm{RU}_j = \mathrm{RU}_j^{\mathrm{CPU}} \tag{9}$$

Therefore, using the server's CPU utilization, we can model the server's power consumption value ΔP_j as follows:

$$\Delta P_j = \begin{cases} P_{j,\mathrm{idle}} + (P_{j,\mathrm{peak}} - P_{j,\mathrm{idle}}) \times UT_j, & \text{if } \mathrm{UT}_j > 0 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Here, the idle power consumption $P_{j,\mathrm{idle}}$ denotes the half of the peak power consumption $P_{j,\mathrm{peak}}$ and UT_j is the percentage of utilization in pm_j. From Eq. (10), it is clear that the server's power consumption is linearly proportional to its CPU usage.

In a virtualized data center, the server's energy consumption is varied over time and depends on the incoming load or the VM's sizes [52].

Corresponding to the workload variation, the CPU usage of a cloud server also varies over time. Thus, we can represent the utilization UT_j as a function of time. Server's total energy consumption ΔE_j is determined via the energy model [50] represented as:

$$\Delta E_j = \int_t \Delta P(\mathrm{UT}_j)\mathrm{d}t. \tag{11}$$

Determining analytical power consumption models for the modern multi-core and multi-thread processors is a very complicated research issue. Therefore, instead of adopting an analytical power model, we take advantage of real-time power consumption data rendered by the SPECpower benchmark results [51]. For this, we picked 2 cloud server configurations of dual-core CPUs as published in Feb 2011: HP ProLiant ML110 G5 (Intel Xeon 3075, (2 cores × 2660 MHz, 4 GB), and HP ProLiant ML110 G4 (Intel Xeon 3040, 2 cores × 1860 MHz, 4 GB). The selected servers' power consumption features are presented in Table 3 [12].

4.2 Overhead of live VM migration

Live VM migration permits to transfer of a VM between the PMs without suspending the running applications and with a short duration of downtime. It also assists in consolidating the VMs to a lesser number of PMs dynamically. Although, live VM migration affects the performance of running applications of migrating VMs [53]. Previous studies have shown that, due to migration, the average degradation of performance along with the downtime of the application running in that VM is almost 10% of its CPU usage [12]. Each migration phase may also produce an SLA violation. Thus, it is a very critical factor to reduce the VM migrations. The live VM migration time primarily relies on the VM's total memory and available network bandwidth. For interpreting the migration model, the overall delay and performance degradation of any VM vm_i [12, 54] can be estimated as:

$$\mathrm{Perf}_{\mathrm{deg}_i} = 0.1 \cdot \int_{t_0}^{t_0+T_{m_i}} \mathrm{UT}_i(t)\mathrm{d}t, \tag{12}$$

$$T_{m_i} = \frac{M_i}{B_i} \tag{13}$$

where $\mathrm{Perf}_{\mathrm{degr}_i}$ represents the performance degradation realized by vm_i ; T_{m_i} indicates the time taken to finish the migration of vm_i ; t_0 denotes the time when the migration of vm_i begins; $\mathrm{UT}_i(t)$ shows the CPU usage of vm_i. M_i shows the total memory utilized by vm_i ; and B_i indicates the available network bandwidth.

Table 3 Servers' energy consumption with respect to load levels in Watts [12]

Server	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP G5	93.7	97	101	105	110	116	121	125	129	133	135
HP G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117

4.3 QoS and SLA violation model

Meeting the customers' QoS requirements is one of the primary objectives of cloud service providers (CSPs). QoS demands are generally determined through SLAs. Furthermore, the excessive VM consolidation may lead to the hotspots in PMs, VMs performance degradation, and increased response time while finishing the jobs executing inside the VMs, which, in result, produces SLA violations and also impacts the QoS performance. In [12], the researchers proposed performance metrics to evaluate the SLA violation levels in an IaaS cloud. The performance metrics are as follows: SLA violation time per active PM (SLATAPM) as determined in Eq. (14), performance degradation due to migrations (PERFDM) represented in Eq. (15), and the merged SLAV metric expressed in Eqs. (16) and (17).

$$\text{SLATAPM} = \frac{1}{m} \sum_{j \in \text{PM}} \frac{T_{\max_j}}{T_{\text{act}_j}} \tag{14}$$

where $T_{\max_j}$ is defined as the total time, when the pm_j has encountered 100 % utilization in any type of d resource and T_{act_j} defines the total active state time for pm_j.

$$\text{PERFDM} = \frac{1}{n} \sum_{i \in \text{VM}} \frac{\text{Perf}_{d_i}}{T_{r_i}} \tag{15}$$

where Perf_{d_i} denotes the performance degradation in vm_i due to migrations and T_{ri} is the total resource demand by vm_i in its lifetime.

As stated by [12], the SLATAPM and PERFDM metrics are both equally essential to characterize the SLA violation level of the infrastructure. The key reason behind the SLATAPM formulation is in the case when a hotspot is created in any PM, or the CPU utilization reaches to 100%, then the running applications' performance inside the VMs assigned to that PM is restricted by the PM's capacity. Consequently, the VMs executing in that PM may not gain their expected performance level throughout the peak workloads. It means the SLATAPM metric value rapidly increases with the PMs' over-utilization. On the other hand, the PERFDM metric is useful to measure the overall performance degradation due to migrations. The value of PERFDM rises with an increase in the VM migrations.

By combining the SLATAPM and PERFDM metrics, we can formulate the SLAV metric to determine the performance degradation and impact of QoS caused due to overloading of PM and also due to migrations. The SLAV metric can be stated as follows:

$$\text{SLAV} = \text{SLATAPM} \times \text{PERFDM} \tag{16}$$

$$\text{SLAV} = \frac{1}{m} \sum_{j \in \text{PM}} \frac{T_{\max_j}}{T_{\text{act}_j}} \times \frac{1}{n} \sum_{i \in \text{VM}} \frac{\text{Perf}_{d_i}}{T_{r_i}} \tag{17}$$

We can minimize the energy consumption at the risk of an increased SLAVs level. Thus, we combined both the SLA violations and Energy factors as ESV performance metric [12], which can be formulated as follows:

$$\mathrm{ESV} = \Delta E \times \mathrm{SLAV} \tag{18}$$

where ΔE signifies the overall energy consumption of all the PMs.

5 Proposed multivariate resource utilization prediction-based hotspots and coldspots mitigation

In this section, our proposed multivariate resource utilization prediction-based hotspots and coldspots mitigation approach is segmented into the following parts: In Sects. 5.1 and 5.2, we discuss the clustering-based stacked BiLSTM deep learning model and feature selection methods, respectively. The proposed algorithms for hotspots and coldspots detection, potential VM selection, and destination PM selection are described in Sect. 5.3.

5.1 Multivariate resource usage prediction

The time series data components can be decomposed into four integral parts: seasonality, trends, level, and noise or randomness. More often, the forecasting in time series applies the information in a time series to predict the given series's future values. A univariate time series contains only one single time-dependent variable. To predict cloud resource usage, the models interpret the temporal usage patterns present in the desired resource metric's historical data and forecast future trends of data. Nevertheless, for robust prediction performance in predicting new values of the target resource metric, the presence of temporal variations in other associated resource metrics can also be considered along with the target resource metric itself. A multivariate time series analysis considers more than one time-dependent variable simultaneously. During forecasting, each variable depends on its past values and has some dependency on other variables of the given dataset. Also, this dependency is utilized for predicting future resource values. Multivariate analysis for resource utilization prediction uses multiple resource usage metrics to investigate the relationships and their consolidated influence on the target resource usage metric. The multivariate time series analysis offers greater statistical power than exploring individual features and strictly model the existence. Each decision includes the study of multiple variables [41]. It applies various statistical techniques to identify the relationship between numerous features and also correlate how an individual metric is more critical while generating a final out-of-sample usage prediction of the target metric.

This paper analyzes the combined effect of different resource usage metrics for future utilization value of the target resource metric. Next, we assume Y is a multivariate time series to describe the system's state by applying other resource metrics y with different time intervals. Here, Y is characterized as:

$$Y = \left(y^1, y^2, ..., y^t, ..., y^T\right) \tag{19}$$

where $y^t = \left[y_1^t, y_2^t, ..., y_L^t\right]^T, \forall = 1, ..., T$.

were T represents total observations used in the analysis, and y_i^t indicates the value of any resource usage metric i, which is reported at t time period. The variable L signifies the number of resource metrics used for observation.

For multivariate time series prediction, the forthcoming value of any resource y_j is predicted using the combination of two individual functions $G_1(.)$ and $G_2(.)$ as:

$$\hat{y}_j^{\,t} = G_1\left(y_j^{t-1}\right) + \sum_{p=1, p\neq j}^{D} G_2\left(y_p^{t-1}\right) \tag{20}$$

where the function $G_1(.)$ is intended to learn the correlation of the target metric with itself, and function $G_2(.)$ is designed to learn the correlation of the target metric with the other associated metrics.

This paper suggests extending the classical univariate time series prediction models to deep learning-based multivariate time series prediction models.

5.1.1 BiLSTM (Bidirectional LSTM)

Bi-LSTM combines LSTM networks, and Bi-directional RNNs (BiRNN) [55], which processes the input sequence in both directions, i.e., forward and backward. It has been shown that the bidirectional networks perform better than the unidirectional LSTM in several domains. Figure 8 shows the unfolded BiLSTM layer architecture comprising a forward and backward layer of LSTM. Here, the forward layer output sequence denoted by $\vec{\mathrm{h}}$, is repetitively computed by applying inputs in a positive order via time series $T - n$ to $T - 1$, while the sequence of backward layer

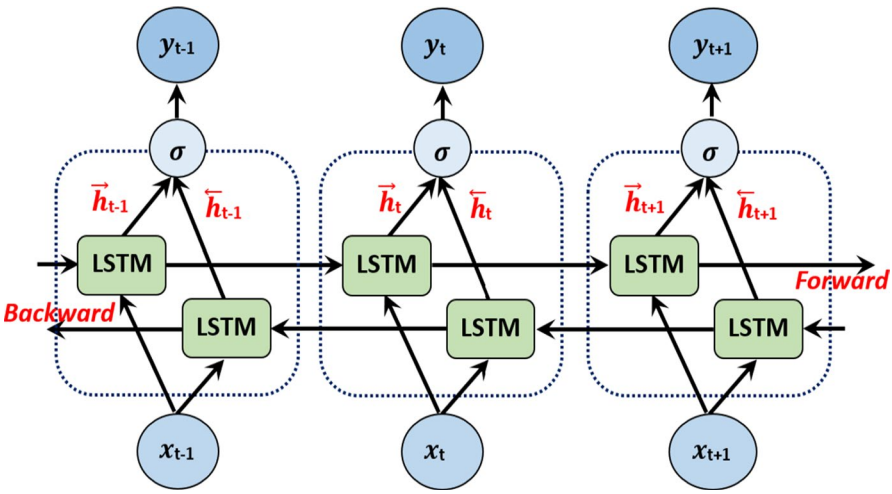


Fig. 8 Structure of BiLSTM

output, $\overleftarrow{h}$, is determined by operating the reversed inputs via time series $T-1$ to $T-n$. By using conventional LSTM equations given in Eqs. (3) to (8), we can find the forward layer and backward layer outputs. Finally, the BiLSTM layer produces Y_T processed output vector, where each element is determined by applying the following equation:

$$y_t = \sigma(\overrightarrow{h}, \overleftarrow{h}) \tag{21}$$

where σ function integrates the two output sequences. Equivalent to the LSTM layer, we can represent the final output of a BiLSTM layer through a vector, $Y_T = [y_{T-n}, ..., y_{T-1}]$, in which the y_{T-1} denotes the predicted next time step value.

5.1.2 Clustering-based stacked bidirectional LSTM

The deep LSTM model can build up with several stacked LSTM networks' hidden layers. The LSTM hidden layer output will be given as an input to the successive LSTM hidden layer. Such stacked layers of architecture improve the capability of neural networks. As discussed earlier, the BiLSTMs utilizes both forward and backward dependencies. Therefore, in this study, we present a Stacked BiLSTM architecture, which uses the BiLSTMs to learn more useful information via the spatial-time series data and the LSTM layer as the top (last) layer of the architecture which only utilize learned features through lower layers, to calculate and generate the predicted values.

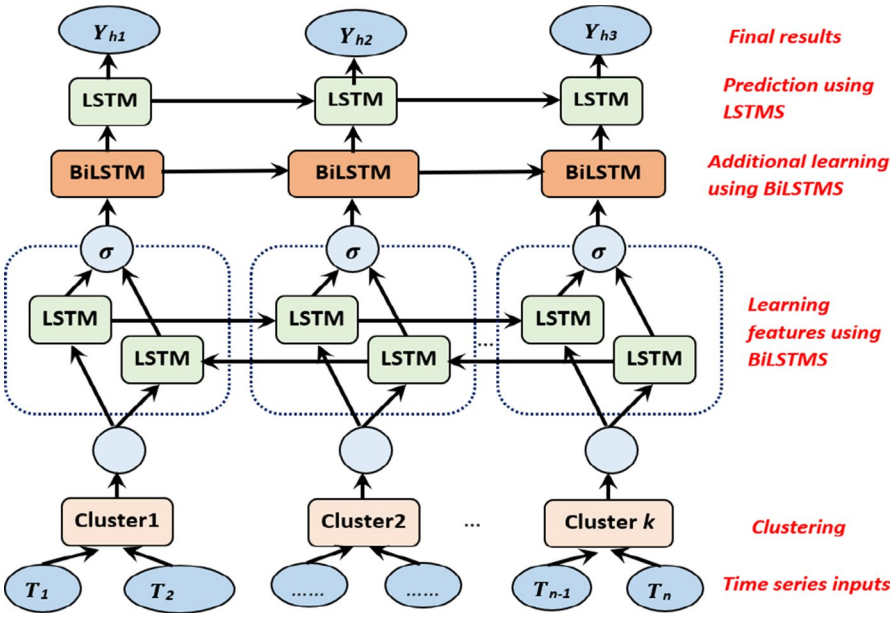


Fig. 9 Proposed architecture

In Fig. 9, we have illustrated the proposed deep learning architecture. In input, it takes the spatial time series data and predicts the future resource utilization values for multiple future time steps. This network is based on the hypothesis that the cluster data contain a group of patterns, and recognizing those groups and training against them, an individual deep learning model for each will result in better predictions.

In Algorithm 1, we have discussed the step-by-step phases of the proposed deep learning network. To calculate the out-of-sample prediction, first, we classify our dataset into training and test sets. Moreover, the training set is classified into k number of clusters with the help of an unsupervised learning model, i.e., K-Means. Obtaining the optimal value of k is a critical factor. If the value of k is very high, then there might be some pattern redundancy, and also, a large number of network models will have to be trained, which would be computationally expensive. While, if the value of k is low then there might be some undetected patterns left in the data and scattered over several clusters. Therefore, the value of k is selected by the trial-and-error method. This model is saved and further used for classifying samples of the test dataset. By using the stacked BiLSTM deep learners, we train each of these clusters separately. Thus, there is a total k number of stacked BiLSTM deep learners for learning of k clustered datasets. For each stacked BiLSTM deep learner, there is a BiLSTM layer that acts as the first feature-learning layer, and, on top of that, an LSTM layer works as the final layer. For complex and exhaustive feature learning using input data, we apply additional BiLSTM layers. The clustering-based stacked BiLSTM deep learners are capable of the multi-step-ahead predictions using the historical resource usage values.

To capture the relevant features set, we combine the clustering-based stacked BiLSTM deep learning model with the feature selection approaches. In the next subsection, we discuss some feature selection schemes for the effective selection of resource metrics from the dataset.

Algorithm 1: Prediction Algorithm

Input: Number of clusters n_c; Multi-variate time series $S_{in} \leftarrow \{T_1, T_2, T_3, \ldots T_n\}$ where $T_i \leftarrow \{x_1, x_2, x_3, \ldots x_{n_f}\}$ and n_f is number of features; Number of iterations N;
Output: New improvised prediction model P_{final}
Divide the time series into two parts:
Training Set $S_{train} \leftarrow \{T_i\}$: i ϵ (0,l) and $l = 0.8 * n$;
Test Set $S_{test} \leftarrow \{T_j\}$: $j \epsilon [l, n]$;
Let **Set** Y_{train} be a set of training outputs/targets
Let **Set** $C \leftarrow \{C_1, C_2, C_3, \ldots C_{n_c}\}$ be a set of clusters where $C_i \leftarrow \{T_i\}$: $T_i \epsilon S_{train}$, and $C_i \cap C_j = \phi$ if i!=j
Let $P_{final} \leftarrow \{P_1, P_2, P_3 \ldots P_{n_c}\}$, where P_i is a weak model.
while (N!=0):
begin
  for X **in** S_{train}:
  begin
    if $X \epsilon C_i$:
    $Y_{h1} \leftarrow h1_i(W1, X)$, where $Y_{h1} = \{y_{11}, y_{12}, y_{13} \ldots y_{1m}\} : m \mathrel{!}= n$
    $Y_{h2} \leftarrow h2_i(Y_{h1})$, where $Y_{h2} = \{y_{21}, y_{22}, y_{23} \ldots y_{2q}\} : q \mathrel{!}= m$
    $Y_{h3} \leftarrow h3_i(Y_{h2})$, where Y_{h3} is the target output.
    Calculate difference between predicted and target value.
    $\varepsilon = Y_{h3} - Y_{target,i}$
    Adjust weights using ε as $\Delta w = \alpha * \varepsilon * h3_i`(Y_{h2})$ $w_{new} = w_{old} + \Delta w$
    Adjust intermediate weights for hidden layers using back-propagation
  $N = N - 1$

5.2 Feature selection schemes for prediction of multivariate resource usages

5.2.1 Pearson's correlation

Pearson's correlation can be defined as the linear correlation between two variables A and B. Therefore, a subtle change in A will also result a similar change in B. This form of relation outputs a value from − 1 to 1, where − 1 represents a negative linear correlation and a + 1 represents a positive linear correlation. Thus, change is proportional. If there is no linear correlation at all, the formula outputs 0. Pearson's correlation is given as:

$$p_r = \mathrm{cor}(a_i, a_j) = \frac{\sum_{n=1}^{N}(a_i^n - \mu_i)(a_j^n - \mu_j)}{\sqrt{\sum_{n=1}^{N}(a_i^n - \mu_i)^2}\sqrt{\sum_{n=1}^{N}(a_j^n - \mu_j)^2}} \tag{22}$$

where A and B are represented as a_i and a_j, respectively. These two variables are resource matrices such that $a_i = \left[a_i^1, a_i^2, a_i^3, \ldots a_i^N\right]$ and $a_j = \left[a_j^1, a_j^2, a_j^3, \ldots a_j^N\right]$ whereas μ_i an μ_j are the mean value of the respective features. Thus, the target metric is correlated with every other resource metric to obtain a value between *-1* and + 1. A threshold is primarily decided, and only those resource metrices are picked whose p_r value greater than the limit of threshold. The threshold limit stated as M, which is the median of all the scores evaluated by the correlation formula.

5.2.2 Spearman's correlation

This correlation is also called as a rank coefficient, is a measure of ranks between two variables (termed as features). This correlation indicates the direction between a_i and a_j. It is assumed that the two variables tend to change at different rates; therefore, they are numbered or ranked in order to achieve a value between − 1 to + 1. Here, + 1 indicates that if one variable increases, the other is bound to increase, and − 1 depicts the correlation that if one variable decreases, so will the other. The values in the metrices are ranked from high to low, meaning the highest value in the variable will be ranked 1 so on and so forth up until the last value is ranked N. The Spearman's correlation coefficient is denoted by r_s and is given by,

$$r_s = 1 - \frac{6\sum_{n=1}^{N}(d^n)^2}{N(N^2-1)} \tag{23}$$

where d^n is the difference between the ranks of the two resource metrices.

5.2.3 Granger causality

Granger's Causality is a statistical procedure which states that if a variable A "Granger causes" another variable B, then past values of A has some valuable details about B that will help predict B in a better way. Here, the two variables are denoted as time series. It is substantial to note that unlike other correlation techniques, Granger causality is asymmetric, meaning if A Granger causes B, it does not mean B will Granger cause A. Let A and B be two-time series denoted by the variables a_i and a_j. The causality is calculated by applying linear regression on the value we need to predict. Suppose we have to predict a_i; therefore, a_i will be formulated in two linear regressions. One, where $\hat{a}_i$ is predicted only using the past values of a_i (termed as $\hat{a}^n_{i\,(\text{restricted})}$) and secondly, a prediction using the past values of a_i as well as a_j (termed as $\hat{a}^n_{i\,(\text{unrestricted})}$).

$$\hat{a}^n_{i(\text{restricted})} = w_0 a_i^{n-1} + w_1 a_i^{n-2} + \cdots + w_{\rho-1} a_i^{n-\rho} \tag{24}$$

$$\begin{aligned}\hat{a}^n_{i(\text{unrestricted})} = & w_0 a_i^{n-1} + w_1 a_i^{n-2} + \cdots + w_{\rho-1} a_i^{n-\rho} + w'_0 a_j^{n-1} \\ & + w'_1 a_j^{n-2} + \cdots + w'_{\rho-1} a_j^{n-\rho}\end{aligned} \tag{25}$$

where w and w' are the weights calculated by the linear regression on learning from the past values. ρ represents the number of steps taken back to predict the next value in the series. G-Test is the test that depicts the overall significance of the calculation and whether or not a_j's past will truly help us predict a_i or not.

$$\text{G-Test} = \frac{\delta_{\text{restricted}} - \delta_{\text{unrestricted}}}{\delta_{\text{unrestricted}}}\left(\frac{N-(2\rho+1)}{\rho}\right)$$

where δ is denoted as sum of square error in prediction. Given as,

$$\delta_{\text{unrestricted}} = \sum_{n=1}^{N} \left(\hat{a}^{n}_{i_{\text{unrestricted}}} - a_i^n\right)^2 \tag{27}$$

$$\delta_{\text{restricted}} = \sum_{n=1}^{N} \left(\hat{a}^{n}_{i_{\text{restricted}}} - a_i^n\right)^2 \tag{28}$$

5.2.4 Mutual information

The mutual information technique measures the relationship between two random variables. It is described as the mutual dependencies between the two variables and how much information a random variable A has about another random variable B. It is given as the two variables' entropy combined subtracted from the sum of the uncertainty of the two variables separately. Here again, the variables are the two resource metrices. One is our candid resource metric (CPU rate), and the other is the resource metric that needs to be validated for acceptance or rejection.

$$E(a_i) = \sum_{n=1}^{N} \rho\left(a_i^n\right) log\left(\rho\left(a_i^n\right)\right) \tag{29}$$

where $\rho\left(a_i^n\right)$ is the probability distribution of the resource metric a_i^n

$$E\left(a_i, a_j\right) = \sum_{n=1}^{N} \sum_{n=1}^{N} \rho\left(a_i^n, a_j^n\right) log\left(\rho\left(a_i^n, a_j^n\right)\right) \tag{30}$$

Therefore, mutual information can be given as

$$M(a_i, a_j) = E(a_i) - E(a_i, a_j) + E(a_j) \tag{31}$$

Larger the value of mutual information depicts a higher dependency of the two resource metrices, and such a metric cannot be omitted. If $M(a_i, a_j)$ is higher than the threshold, then it is accepted by the algorithm. The range of the value is $[0, \infty]$.

5.2.5 ANOVA

Analysis of variance is the ratio between variability between the sample means and variability within the distributions.

$$F = \frac{\text{Variance Between}}{\text{Variance Within}} \tag{32}$$

$$\text{Variance Between} + \text{Variance Within} = \text{Total Variance} \tag{33}$$

The variability between the means in the numerator is large compared to the variance within the samples in the denominator, the ratio will be much larger than 1. Then, the samples most likely do not come from a common population. Therefore,

we discard the null hypothesis, which states that the means are equal. The inputs will be taken in the format of (nsamples, nfeatures) and output a 1D array, including an F test score for each of the feature (1, nfeatures).

5.2.6 Kendall's Tau

Kendall's nonparametric Tau, more commonly known as Kendall's rank correlation coefficient, is the degree of consistency between two columns of ranked data. Kendall's Tau outputs a value from − 1 to + 1. This method deals with concordant and discordant pairs in two columns. A concordant pair represents the measure ranks lie below a certain rank that is greater than it. Therefore, a discordant pair shows the observed ranks lower than a stated rank whose values are lesser than the stated rank.

5.3 Prediction-based hotspots and coldspots mitigation

Prediction-based hotspots and coldspots mitigation perform 3 steps to accomplish an optimal dynamic VM consolidation (as shown in Algorithm 2). First, the algorithm distinguishes the PMs having hotspots (Sect. 5.3.1), then selects some VMs from each hotspot PM to perform migration (Sect. 5.3.3) and determines destination PMs for the VMs (Sect. 5.3.4). To detect coldspots (Sect. 5.3.2), the algorithm chooses one PM with the lowest CPU usage from the PM list not having hotspots and migrates its deployed VMs to a suitable destination PM.

Algorithm 2: Dynamic hotspots and coldspots mitigation

```
Input: PM_active
Output: MigrationPlan
for each pm ϵ PM_active do
begin
    if(HotspotsDetection(pm)=true) then
    SelectedVM ← VM with minimum EQV;
    MigrateVMList ← pm.SelectedVM
    PM_hot ← PM_hot ∪ pm
end
for each pm not in PM_hot do
begin
    if(ColdspotsDetection(pm)=true) then
    PM_cold ← PM_cold ∪ pm
end
MigrateVMList ← All VMs of PM_cold;
PM_available=PM_active - PM_hot - PM_cold;
MigrationPlan ← Destination PMs from PM_available for MigrateVMList;
return MigrationPlan
```

5.3.1 Prediction-based hotspots detection

In the process of dynamic VM consolidation, each PM must be identified whether having a hotspot or not. To detect the PMs with hotspots, we propose a

prediction-based hotspots detection method as presented in Algorithm 3. It takes a set of active PMs as input and determines whether the PMs creating a hotspot or not. Algorithm 3 is explained step-by-step as follows: In line 3, we set the minimum window length of historical data as 20 using the cross-validation technique for best time window selection. In line 3, the utilization of a pm at time instant t is determined by using the current usage and total capacity of pm. For hotspot detection, we use a dynamic upper threshold method. If the utilization value of at least one resource (i.e., memory or CPU) for any pm is greater than the threshold value, then it has a hotspot. In this method, the threshold value is decided by using the median absolute deviation (MAD) as discussed in [12]. In lines 6–10, if the PMs have insufficient utilization data, then the hotspot decision is only made by comparing the current utilization of pm $\mathrm{UT}_{\mathrm{pm}}(t)$ with a static threshold. In lines 12–16, if the PMs have sufficient history data, then the algorithm uses the future value as well. For this, it first determines the future utilization of pm by applying the clustering-based stacked BiLSTM model (line 12). The pm is considered to be in the set of hotspots pm if the current and future usage value is higher than the upper threshold level of pm (line 13). This condition demonstrates that the pm is a potential candidate that executes the migration operation if and only if it has a hotspot in both the current time instant t and near future at time instant $t + 1$. It means Algorithm 3 analyzes not only the present situation but also the near-future situation. It can avoid useless migrations by performing necessary migration only. Moreover, a clustering-based stacked BiLSTM deep learning model can avoid an SLA violation rate by detecting a pm has a hotspot in advance.

Algorithm 3: Prediction Based Hotspots Detection

1 **Input**: PM_{active}
2 **Output**: Is PM having hotspot or not
3 $windowLength \leftarrow 20$; $UT_{pm}(t) = CU_{pm}(t)/Cap_{pm}(t)$; Set Th_{hot} applying MAD = 1-s*Mad;
4 **for each** $pm \epsilon PM_{active}$ **do**
5 **begin**
6 **if** $Length.UTHistory_{pm}(t) < windowLength$ **then**
7 **if**$(UT_{pm}(t) > Th_{static})$ **then**
8 **return** true;
9 **else**
10 **return** false;
11 **else**
12 predict $UT_{pm}(t + 1)$ using clustering-based stacked BiLSTM model;
13 **if**$((UT_{pm}(t) > Th_{hot})$ && $(UT_{pm}(t + 1) > Th_{hot}))$ **then**
14 **return** true;
15 **else**
16 **return** false;
17 **end**

5.3.2 Prediction-based coldspots detection

Once the hotspots are identified, the coldspots detection procedure is executed. To bring down the overall energy consumption, the actively used PMs are

switched into a low-power state or sleep state. The proposed prediction-based coldspots detection algorithm works as follows: Algorithm 4 takes the active PM set's input and evaluates whether the PM is having a coldspot or not. The phases of Algorithm 4 is almost similar to the steps of Algorithm 3. The differences lie in lines 7 and 13. If the $\mathrm{UT}_{\mathrm{pm}}(t)$ is less than or equal to $\mathrm{Th}_{\mathrm{cold}}$, then the pm is having coldspot. After collecting the windowLength of 20 utilization historical data, the pm is considered to be in the set of coldspots pm if the current and future utilization values are less than or equal to the lower threshold.

Algorithm 4: Prediction Based Coldspot Detection

1 **Input**: PM_{active}
2 **Output**: Is PM having coldspot or not
3 $windowLength \leftarrow 20$; $UT_{pm}(t) = CU_{pm}(t)/Cap_{pm}(t)$; Set $Th_{cold} = 30\%$
4 **for each** $pm \epsilon PM_{active}$ **do**
5 **begin**
6 **if** $Length.UTHistory_{pm}(t) < windowLength$ **then**
7 **if**$(UT_{pm}(t) \leq Th_{cold})$ **then**
8 **return** true;
9 **else**
10 **return** false;
11 **else**
12 predict $UT_{pm}(t+1)$ using stacked BiLSTM model;
13 **if**$((UT_{pm}(t) \leq Th_{cold})$ && $(UT_{pm}(t+1) \leq Th_{cold}))$ **then**
14 **return** true;
15 **else**
16 **return** false;
17 **end**

5.3.3 Energy and quality aware VM selection

After detecting the PMs with hotspots, the next stage is to pick VMs to be migrated from that PM. It is obvious that the selection of VMs with lower CPU utilization will lead to a smaller rise in energy consumption. Thus, to improve the QoS level and migration time, we should select the VMs with smaller memory for migration. Therefore, whenever any PM creates hotspot, we compute the cost of energy and QoS (EQV) for each of the VMs assigned to it. The EQV value of VM stated as follows:

$$\mathrm{EQV}_{\mathrm{vm}} = w1 \cdot \frac{\mathrm{CU}_{\mathrm{vm}}^{\mathrm{CPU}}(t)}{\mathrm{Cap}_{\mathrm{vm}}^{\mathrm{CPU}}(t)} + w1 \cdot \frac{\mathrm{CU}_{\mathrm{vm}}^{\mathrm{MEM}}(t)}{\mathrm{Cap}_{\mathrm{vm}}^{\mathrm{MEM}}(t)} \tag{34}$$

In comparison to a VM with a higher EQV cost, we can achieve maximum energy-efficiency and minimum performance degradation by migrating the VM with a smaller EQV cost.

5.3.4 Destination PM selection

After building the list of selected VMs for migration, VM consolidation's last step is to obtain the target PMs for the selected VMs. VMs to PMs placement is an NP-hard problem and could be solved with the help of a multidimensional bin packing problem. In the bin packing, each PM is considered as a bin and each VM is mapped with an item of a different volume. The bin packing problem's objective is to pack all the items into bins such that the used number of bins can be minimized [13]. To achieve this goal, we designed a solution based on the BFD algorithm because it used a minimum number of bins compared to the First-Fit and Next-Fit schemes [56]. The step-by-step process of destination PM selection is illustrated in Algorithm 5.

Algorithm 5: Destination PM Selection

Input: $PM_{List}, VMList$
Output: Assignment of VMs
Set $windowLength \leftarrow 20$;
if($Length.UTHistory_{pm}(t) < windowLength$) **then**
return PBFD($PM_{List}, VMList$);
else
Set $DestinationPM \leftarrow \phi$;
Set $MigrationList \leftarrow \phi$;
SortedVMList ← Sort VMs of $VMList$ by their CPU usage prediction in desceneding order;
for each $vm \epsilon VMList$ **do**
begin
  $Allocation \leftarrow false$;
  for each $pm \epsilon PMList$ **do**
  begin
    if($(UT_{pm}(t) + UT_{vm}(t) \leq Cap_{pm}(t))$ && $(UT_{pm}(t+1) > maxUT)$) **then**
    DestinationPM ← $DestinationPM \bigcup pm$;
    $Allocation \leftarrow true$;
  end
  if($Allocation = true$) **then**
    $MigrationList = MigrationList \bigcup \{vm, DestinationPM\}$;
end
return $MigrationList$;

The destination PM selection algorithm sorts all chosen VMs in the decreasing sequence of their predicted CPU value. In next step, the algorithm assigns each VM to a PM having maximum CPU utilization after placement. This is because selecting the PM with higher CPU usage fulfills the consolidation goal effectively, which allows the packing of many VMs to a lesser number of PMs. In the initial phase, when the length of historical records is less than the window length, we apply the Power-aware BFD (PBFD) algorithm [12] as no predicted CPU utilization value is available yet.

5.4 Complexity analysis

To compute the time complexity of Prediction Algorithm (Algorithm 1), we observe the computational complexities of different neural network approaches analyzed in the work of Behera et al. [57]. Based on this study, we consider a basic

neural network model having Q number of cells in the hidden layer, T inputs, and W outputs. In the case of simple multi-layer perceptron (MLP) with one hidden layer, we can formulate the total parameters as: $C_{\mathrm{MLP}} = \mathrm{TQ} + \mathrm{QW}$. For RNN, if we replace the hidden layer cells with RNN units, then the total parameters will be $C_{\mathrm{RNN}} = \mathrm{TQ} + Q^2 + \mathrm{QW}$, here Q^2 represents the recurrent connection. Similarly, in the case of LSTM cell with 3 gates and a cell state, the total parameters can be represented as $C_{\mathrm{LSTM}} = 4\mathrm{TQ} + 4Q^2 + 3Q + \mathrm{QW}$. While in the case of a GRU cell with two gates and a hidden state, the total parameters can be expressed as: $C_{\mathrm{GRU}} = 3\mathrm{TQ} + 3Q^2 + 3Q + \mathrm{QW}$. In the case of BiLSTM, we know that each BiLSTM layer is stacked with two BiLSTM layers. Thus, we can express the total parameters for a BiLSTM layer as: $C_{\mathrm{BiLSTM}} = 2 * C_{\mathrm{LSTM}}$. If we stack up P number of layers, the total parameters can be represented as: $C_{\mathrm{BiLSTM}} = P * C_{\mathrm{LSTM}}$. If we consider the input layer to be the same across all stacked networks, then the overall complexity of the proposed prediction model with Θ number of training epochs can be expressed as $C_{\mathrm{model}} \approx O((Q(Q + W) * \Theta)$.

In order to describe the time complexity of hotspots and coldspots mitigation algorithm, we analyze the time complexity of its sub-algorithms, respectively. The hotspots and coldspots mitigation algorithm (Algorithm 2) comprises Prediction-Based Hotspots Detection (Algorithm 3), Prediction-Based Coldspot Detection (Algorithm 4), and Destination PM Selection (Algorithm 5). Assuming s number of PMs and k number of VMs, the time complexity of hotspots and coldspots mitigation algorithm is $O(sk)$ in worst-case. The time complexity of both Prediction-Based Hotspots Detection (Algorithm 3), and Prediction-Based Coldspot Detection (Algorithm 4) is $O(k)$ in worst-case. Destination PM Selection (Algorithm 5) identifies each PM for MigrateVMList, so its worst-case time complexity is $O(sk)$. Thus, the total time complexity of hotspots and coldspots mitigation algorithm (Algorithm 2) is $O(sk)$.

6 Experimental results

First, we discuss workload data, the simulation environment with assumptions, performance metrics, and benchmark approaches. Then, we present the experimental evaluations for validation of our proposed method.

6.1 Workload data

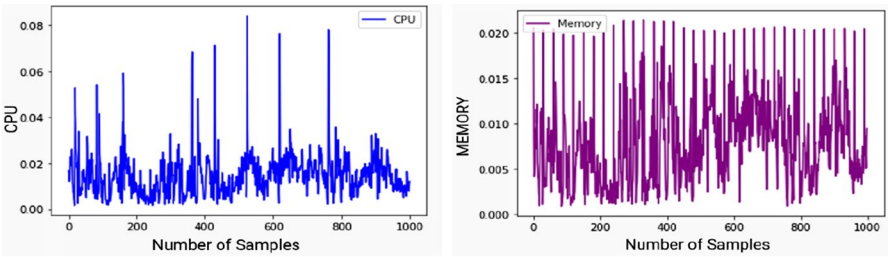
In this work, we apply the real trace data of a Google cluster [39]. This trace provides cluster's workload data containing 12,500 PMs recorded in May 2011. The Google cluster trace comprises running-time traces of over 650 thousand real-time jobs dynamically entering, executing, and leaving the Google cluster for 29 days, where resource utilization values are bundled at 10 seconds time-frame. Resource tables of Google cluster trace are broadly classified into three major categories, *Machines, Jobs & Tasks, and Resource Usage*. Each category has one or more than one table containing multiple features. In the present study, we have used the resource usage

Table 4 List of resource usage metrics

Resource metrics	Description
CPU	Usage of aggregated CPU
MEM	Usage of aggregated memory
MAXM	Maximum memory utilization observed
VM	Assigned memory (may or may not be used)
MAXC	Maximum CPU utilization (over interval)
UPC	Aggregate unmapped page cache
TPC	Usage of total page cache
CPI	Cycles per instruction over all nodes
MAXD	Maximum disk I/O time detected (over interval)
DSP	Disk capacity space utilization over all nodes
MAI	Memory accesses per instruction over all nodes
DIO	Disk I/O time over all disks

table, and from a total of 20 columns, only 12 different resource usage metrics are taken for CPU and memory usage prediction. Like the others, resource metrics such as start_time, end_time, job_ID, and tasks_ID, etc., do not provide any correlation with CPU or memory. The diverse resource utilization metrics applied in this work are described in Table 4.

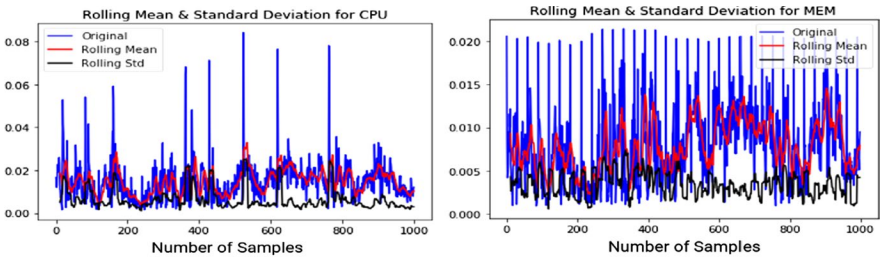
During the data preparation, the rows with missing and incomplete information are dropped. Each row corresponds to data collected by the cluster for over 10 seconds. We have merged rows corresponding to 5 minutes' worth of data into a single row for prediction. A total of 10 days of resource workload with 86,880 samples is taken to train the deep learning models and predict CPU and memory usage. Both resource usage values are normalized, respectively. In Figures and , we represent the sample CPU and memory data. To show the characteristics of the workload traces, in Figures and , we present the rolling mean and standard deviation of CPU and memory data, respectively. Furthermore, we have applied different feature selection approaches to decide the most relevant set of features for prediction. For evaluation, we categorized the data into a validation, training, and test sets. For validation, we



(a) Sample of CPU values

(b) Sample of memory values

Fig. 10 Sample values of Google cluster traces



(a) Mean and standard deviation for CPU **(b)** Mean and standard deviation for memory

Fig. 11 Analysis of rolling mean and standard deviation

perform the parameter tuning of different models. Then, we select the best fitting parameters for each of the corresponding models to generate future predictions for the next 90, 180, and 270 steps (Figs. 10 and 11).

6.2 Simulation environment

To validate the proposed method's performance, we use CloudSim 3.0.3 toolkit [17]. We have conducted our simulations on a compute server powered with Intel(R) Xeon(R) Processor E5-2630 v3 @ 2.40 GHz with 2 CPU sockets ($2 * 8 = 16$ core), 1 TB Hard disk, 128 GB RAM, and 64-bit Windows 10 OS. Moreover, a cloud data center is simulated with 800 heterogeneous PMs. Each PM of data center comprises a 2-core processor, 4 GB memory, 1GB/s network bandwidth, and MIPS is randomly generated between 1860 or 2660 MIPS for each core. In the PM list, half of the PMs are HP ProLiant ML110 G4 servers while the other half are HP ProLiant ML110 G5 servers. Based on the information available in Amazon EC2, 4 VM instances are used, i.e., small (1000 MIPS, 1.7 GB), micro (500 MIPS, 613 MB), High-CPU Medium (2500 MIPS, 0.85 GB) and Extra large (2000 MIPS, 3.75 GB)

Table 5 Hyper-parameter settings of the proposed model

Architecture details	
Parameter	Value
Default hidden layers	3
Units in hidden layers	60,120,60
Training step/epochs	70
Batch size	100
Loss function	Mean squared error
Optimizer	Adam
Activation function	tanh
Input layer neurons	12
BiLSTM layer neurons	60,120,60
Final layer neurons	1

[12]. The energy consumption value of all PMs is computed using the data represented in Table 3.

The list of hyper-parameters for the proposed deep learning model is shown in Table 5. To implement deep learning models' network architecture, we have used the Python language with the Keras library through the TensorFlow back-end via functional API.

6.3 Evaluation metrics

We have applied different performance metrics to validate the effectiveness of the proposed hotspots and coldspots mitigation approach. For analyzing the accuracy of prediction models, we have used the RMSE: Root Mean Square Error metric, which computes the standard deviation of the absolute errors as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{P}(y_i^t - \hat{y}_i^{\,t})^2}{\Delta T}} \tag{35}$$

where ΔT indicates the length of prediction at 90, 180, and 270 steps. We use i to represent the desired resource usage metric. y_i^t shows the actual resource usage metric's value at particular time instant t, and $\hat{y}_i^{\,t}$ indicates the predicted value of resource metric at time instant t. We have used the iterative multi-step ahead resource usage predictions, which predict the CPU and memory resource utilization at 90, 180, and 270 steps ahead.

Then, we analyze the overall energy consumption across all PMs of data center. Each PM's energy consumption value is computed via the power and energy model, as discussed in Sect. 4.1. Next, we have considered the total migrations of all VMs during the mitigation of hotspots and coldspots in the data center. Another metric we have utilized to monitor the QoS and SLA violations. For this, we calculated the SLATAPM, PERFDM, and SLAV metrics, as explained in Sect. 4.3. Finally, the ESV metric is applied to measure the trade-off between QoS and energy consumption, as expressed by Eq. (18).

6.4 Benchmark approaches

For overall performance analysis of the proposed scheme, we have implemented the following algorithms present in the literature:

1. *Sercon* [58] This technique is based on the greedy bin-packing algorithm, which follows all-or-none migration property. In order to free the least loaded host, it tries to migrate all the VMs from the least loaded host to the most loaded host. During the migration process, if any of them is failed, then no migration is performed.
2. *ACSVMC (Ant Colony System based VM Consolidation)* [20] ACSVMC policy follows the meta-heuristic bin-packing approach. It applies an ant colony algo-

rithm to minimize SLA violations and energy consumption and produces a migration plan.

3. *MBFD (Modified Best Fit Decreasing)* [21, 50] In MBFD, the VMs are assigned to a most-loaded PM only if the total utilized capacity of the destination PM and migrated VM is not violating the hot threshold limit of the PM's total resource capacity. It means:

$$\mathrm{UT}_{\mathrm{pm}}(t) + \mathrm{UT}_{\mathrm{vm}}(t) \leq \mathrm{Th}_{\mathrm{hot}} \times \mathrm{Cap}_{\mathrm{pm}}(t) \tag{36}$$

Although the MBFD algorithm does not apply any prediction model, thus it can only migrate VMs of the PMs, creating hotspots to avoid SLA violations. Moreover, MBFD maximizes energy-efficiency via mitigating coldspots.

4. *Modified First Fit Decreasing (MFFD)* [21]

The MFFD algorithm applies the First Fit (FF) strategy and allocates a VM to the first PM that meets the current resource demands of the VM and ensures that VM and PM's total utilized capacity is less than the hot threshold limit of the PM's full resource vector. It means:

$$\mathrm{UT}_{\mathrm{pm}}(t) + \mathrm{UT}_{\mathrm{vm}}(t) \leq \mathrm{Th}_{\mathrm{hot}} \times \mathrm{Cap}_{\mathrm{pm}}(t) \tag{37}$$

5. *PUP-VMC (PM Usage Prediction-aware VM Consolidation)* [21] This technique only inspects the future resource usage of a target PM for assigning any VM. It also ensures that the PM's predicted total utilized capacity and the VM's current demand capacity should be lesser than the hot threshold limit of the PM's total resource capacity. It means:

$$\mathrm{UT}_{\mathrm{pm}}(t+1) + \mathrm{UT}_{\mathrm{vm}}(t) \leq \mathrm{Th}_{hot} \times \mathrm{Cap}_{\mathrm{pm}}(t) \tag{38}$$

6.5 Simulation results

1. *Prediction performance using all features* We assess multivariate prediction models' accuracy for predicting future CPU usage and memory usage in Google

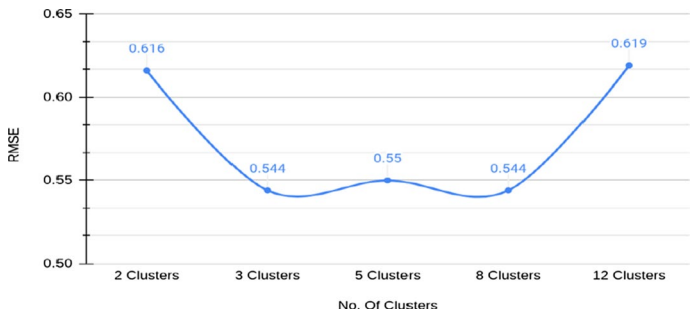


Fig. 12 Cluster analysis

Table 6 RMSE analysis for CPU and memory usage prediction using Google cluster's trace

RMSE of CPU usage prediction									
Model	Univariate			Bivariate			All features		
	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps
ARIMA	0.00667	0.00674	0.0064	0.01087	0.01113	0.01122	0.00762	0.00773	0.00777
Linear Regression	0.00567	0.00571	0.00572	0.00555	0.00563	0.00568	0.00549	0.00555	0.00557
GRU	0.00560	0.00564	0.00566	0.00549	0.00553	0.00554	0.00539	0.00544	0.00547
LSTM	0.00558	0.00562	0.00565	0.00546	0.00550	0.00552	0.00537	0.00541	0.00542
BiLSTM	0.00555	0.00559	0.00561	0.00542	0.00546	0.00548	0.00534	0.00538	0.00540
Clustering-based Stacked BiLSTM	**0.00551**	**0.00555**	**0.00558**	**0.00540**	**0.00543**	**0.00546**	**0.00531**	**0.00535**	**0.00537**
RMSE of memory usage prediction									
Model	Univariate			Bivariate			All features		
	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps
ARIMA	0.00542	0.00546	0.00544	0.00545	0.00556	0.00546	0.00545	0.00556	0.00546
Linear Regression	0.00475	0.00475	0.00476	0.00473	0.00473	0.00473	0.00424	0.00424	0.00425
GRU	0.00456	0.00457	0.00457	0.00456	0.00456	0.00457	0.00413	0.00415	0.00417
LSTM	0.00452	0.00452	0.00453	0.00455	0.00455	0.00455	0.00411	0.00413	0.00414
BiLSTM	0.00450	0.00450	0.00451	0.00451	0.00451	0.00451	0.00409	0.00411	0.00412
Clustering-based Stacked BiLSTM	**0.00447**	**0.00448**	**0.00450**	**0.00444**	**0.00445**	**0.00445**	**0.00405**	**0.00407**	**0.00409**

Here, the entries in bold shows the best performance of model

cluster workloads. In Fig. 12, we analyze the optimal size of clusters to be utilized for the proposed deep learning network model. With 3 clusters, the overall RMSE is minimal; hence, we set the cluster size to 3. The influence of all features (as discussed through Table 4) is investigated for out-of-sample CPU usage and memory usage predictions at 90 steps (15 min), 180 steps (30 min), and 270-steps-ahead (45 min). In Table 6, we present the RMSE of the multi-step ahead CPU and memory utilization predictions caused employing univariate, bivariate, and all features. From the table, it can be observed that the error produced by the multivariate models with all features is smaller than the error of univariate models. It is due to the fact that the multivariate methods analyze the correlations between the desired resource metric, and therefore, they can acquire the overall dynamics of the time series data. Moreover, the proposed prediction method performed bet-

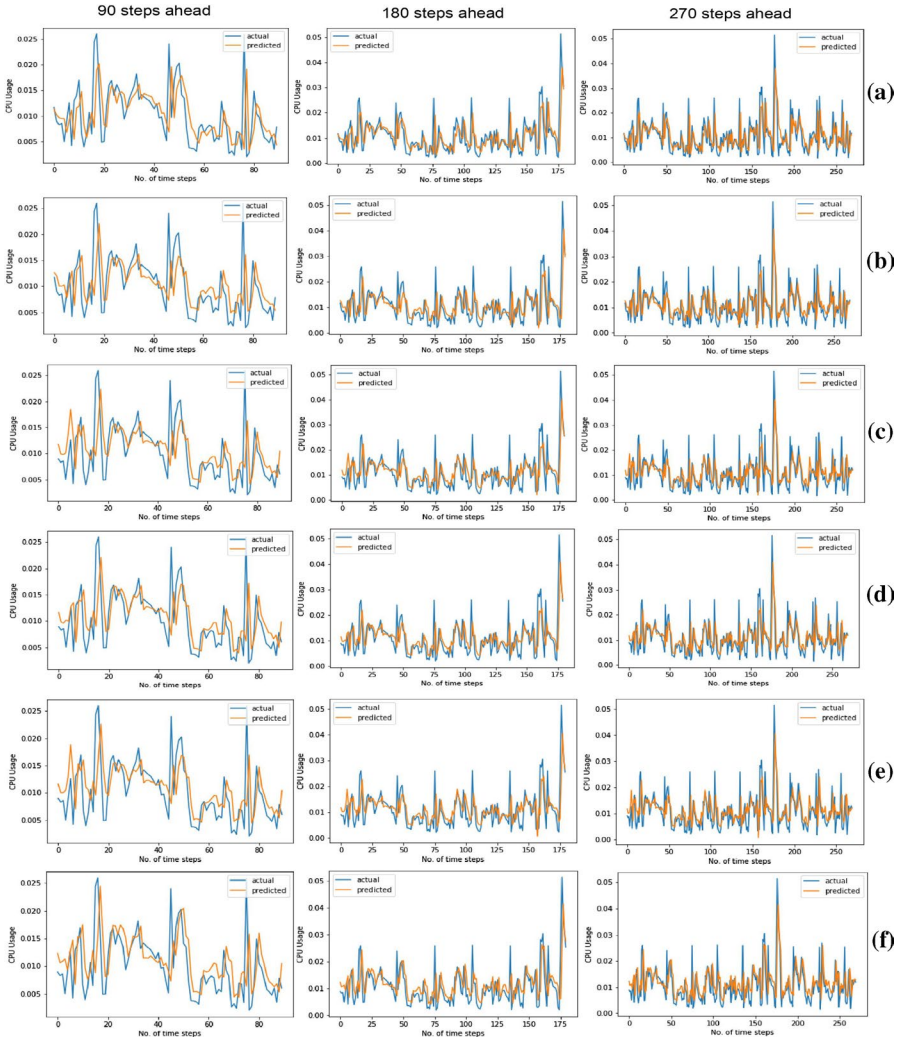


Fig. 13 Multi-steps-ahead CPU prediction with all features using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

ter than the other prediction models and generated minimum RMSE in all cases. Furthermore, in Figs. 13 and 14, we present the predictions of ARIMA, Linear regression, GRU, LSTM, BiLSTM, and proposed method for CPU and memory usage predictions, respectively, using all 12 features.

2. *Prediction performance using feature selection techniques* Figure 15 analyzes the Pearson and Spearman correlation schemes, Mutual information, Granger's causality, ANOVA, and Kendall's feature selection methods at different threshold values to distinguish the most suitable features from the given set. In the Pearson (Fig. 15a), we can see that features such as DIO, MAXC, MAXD, CPI, and MAI

Fig. 14 Multi-steps-ahead memory prediction with all features using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

are clearly below the threshold and must not be selected as the prime features for training the model to predict the CPU usage. In the Spearman (Fig. 15b), values below the median rank such as MAXM, DIO, CPI, MAI are omitted from our selected features. In the Granger causality (Fig. 15c), after calculating the G-Test value for each of the resource metrics against that of our candid metric (CPU Rate), we may omit any feature whose G-Test value is lower than that of the threshold. In the Mutual information (Fig. 15d), values below the Median rate of mutual information are suggested to be removed for optimum result. In the ANOVA (Fig. 15e), values below the median rank such as MAXM, DIO, DSP,

Table 7 Selection of features using different feature selection methods

Method	Set of features										
	VM	MEM	UPC	MAXM	TPC	DIO	MAXC	DSP	MAI	CPI	MAXD
Kendall	✓	✓	✓	×	✓	×	✓	✓	×	×	✓
ANOVA	✓	✓	✓	×	✓	×	✓	×	×	×	✓
Pearson	✓	✓	✓	✓	✓	×	×	✓	×	×	×
Spearman	✓	✓	✓	×	✓	×	✓	✓	×	×	✓
Granger causality	×	✓	✓	×	✓	✓	×	✓	✓	×	×
Mutual information	✓	✓	✓	×	✓	✓	×	✓	✓	✓	×

CPI, and MAI are eliminated from the feature set. In Kendall's Tau (Fig. 15f), we have tested the feature selection with the nonparametric Tau B. Many of the methods we have encountered have clearly marked CPI and MAI as features with the least correlation to CPU utilization.

Table 7 presents the selection of features assuming a median M as a threshold to compare the various feature selection methods. A "Yes" entry in the table represents the selection of particular features, whereas a "No" entry shows the feature is not selected by the method. From the tabular results, it can be noted that applying Spearman, Kendall, and Pearson feature selection techniques produce almost an identical set of feature preferences. In Table 8, we represent the RMSE of all feature selection methods for multi-step ahead CPU and memory usage predictions. From the results, it can be perceived that multivariate models with a selected set of features deliver better performance than the models with all sets of features. The tabular results report that the feature selection approaches achieve better performance with other resource utilization prediction models. For example, CPU utilization prediction, ARIMA, and linear regression models perform better with the Kendall feature selection technique, while the GRU, LSTM, BiLSTM, and proposed clustering-based stacked BiLSTM model work better with Spearman correlation-based feature selection technique. Form memory usage prediction ARIMA model performs better with the Pearson feature selection technique. In comparison, the other models work better with Spearman correlation-based feature selection approach. Figures 16, 17, 18, 19, 20 and 21 depict resource usage prediction models' performance using the Pearson, Spearman, and Kendal methods, respectively, for different multivariate prediction models. We compare the prediction performance at a different step size of 90, 180, and 270 steps ahead. From the results, it is noted that with the increase in step-size, all methods' predictions degrade. It is due to the errors' accumulation for executing multiple steps ahead of out-of-sample predictions. It is also observed that the prediction techniques generate different predictions with different feature selection methods.

3. *SLA violations* In this experiment, we simulated many VMs and obtained the plots on different threshold values set within 30% to 100%. Figure 22a shows the SLA violations provoked by Sercon, MFFD, PUP-VMC, MBFD, ACS-VMC, and proposed methods at different threshold values. As a result, the proposed method

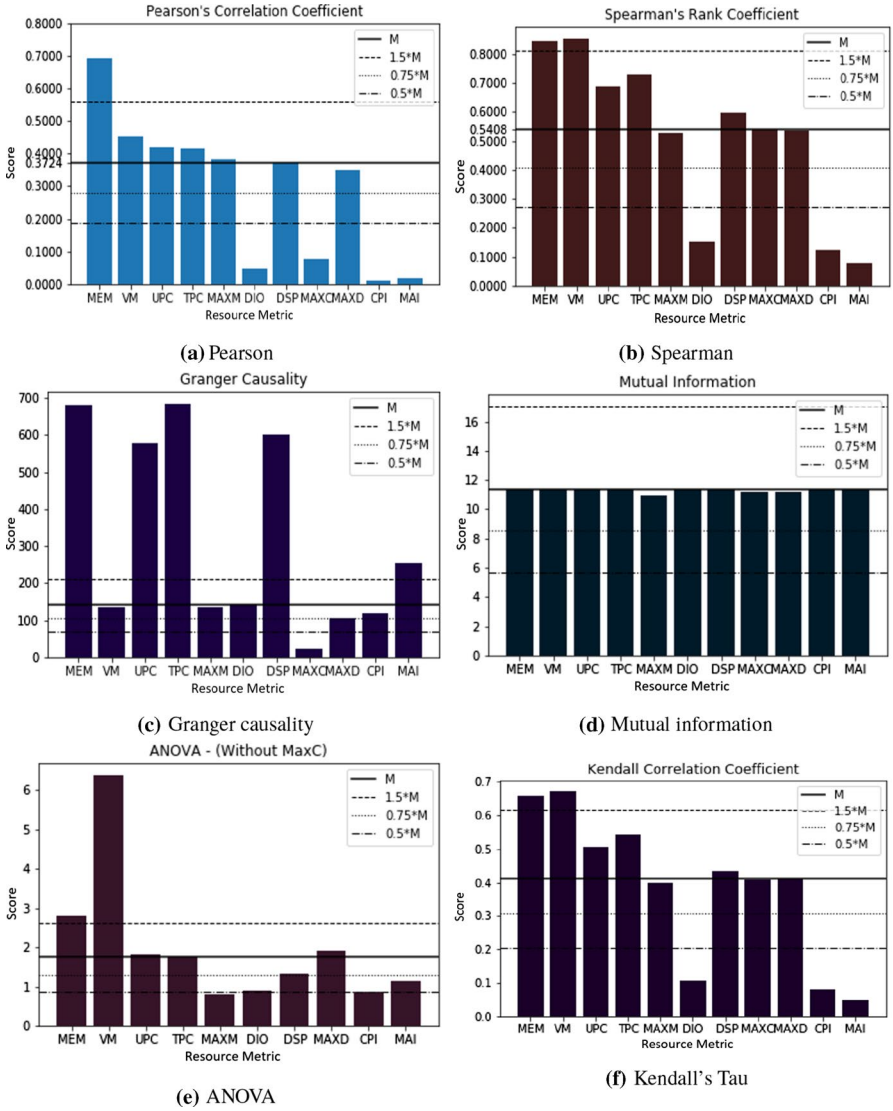
Table 8 RMSE of CPU and memory usage prediction for selected features in Google cluster trace

RMSE of CPU usage prediction									
Model	Pearson			Spearman			Kendall		
	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps
ARIMA	0.00763	0.00774	0.00779	0.00763	0.00774	0.00778	**0.00761**	**0.00773**	**0.00776**
Linear Regression	0.00540	0.00544	0.00546	0.00539	0.00543	0.00549	**0.00538**	**0.00541**	**0.00546**
GRU	0.00536	0.00540	0.00543	**0.00535**	**0.00540**	**0.00542**	0.00535	0.00540	0.00543
LSTM	0.00535	0.00539	0.00541	**0.00533**	**0.00538**	**0.00540**	0.00534	0.00538	0.00540
BiLSTM	0.00534	0.00539	0.00541	**0.00532**	**0.00536**	**0.00538**	0.00533	0.00537	0.00539
Clustering-based stacked BiLSTM	0.00533	0.00537	0.00539	**0.00530**	**0.00535**	**0.00536**	0.00531	0.00536	0.00537
RMSE of memory usage prediction									
Model	Pearson			Spearman			Kendall		
	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps	90 steps	180 steps	270 steps
ARIMA	**0.00488**	**0.00488**	**0.00485**	0.00545	0.00546	0.00546	0.00535	0.00535	0.00536
Linear Regression	0.00434	0.00435	0.00435	**0.00430**	**0.00431**	**0.00431**	0.00431	0.00432	0.00432
GRU	0.00433	0.00434	0.00434	**0.00429**	**0.00428**	**0.00427**	0.00430	0.00430	0.00430
LSTM	0.00429	0.00430	0.00430	**0.00423**	**0.00424**	**0.00425**	0.00425	0.00425	0.00426
BiLSTM	0.00428	0.00429	0.00429	**0.00422**	**0.00423**	**0.00424**	0.00424	0.00424	0.00425
Clustering-based stacked BiLSTM	0.00426	0.00427	0.00428	**0.00422**	**0.00423**	**0.00423**	0.00423	0.00424	0.00424

The bold entries represents the best performance of feature selection method in various networks

leads to fewer violations than the other existing approaches. The reason behind the high performance is that the proposed method applies prediction and assures that the destination PM does not create any hotspot whenever any VM is migrated on it. To avoid further SLA violations, the proposed method always tries to migrate VMs from the PMs that may create hotspots and predicted hotspots only. Thus, it prevents 100% usage of PMs and prevents the migration of VM instances to PMs, showing the potential of turning into hotspots by predicting the future utilization of PMs and VMs through the proposed prediction method.

4. *Energy consumption analysis* PMs' energy consumption relies on the utilization of resources mainly memory, network bandwidth, CPU, etc. However, existing researches reveal that the CPU utilizes more power than other resources. Hence, we represent the PM's resource usage by its CPU usage. As compared to the



(a) Pearson (b) Spearman (c) Granger causality (d) Mutual information (e) ANOVA (f) Kendall's Tau

Fig. 15 Feature selection with **a** Pearson's correlation coefficient **b** Spearman's correlation **c** Granger's causality **d** Mutual information **e** ANOVA and **f** Kendall's Tau

energy consumption caused by different algorithms, the suggested approach uses a lesser amount of energy, as represented in Fig. 22b. The reason being that the PMs having their usage lower than the threshold limit are switched to power-saving mode. The energy consumption of PMs is further optimized via packing the VMs into the most loaded PMs.

5. *Migration cost analysis* As displayed in Fig. 23a, the total migrations are less when compared with the existing algorithms. The depreciation of migrations due

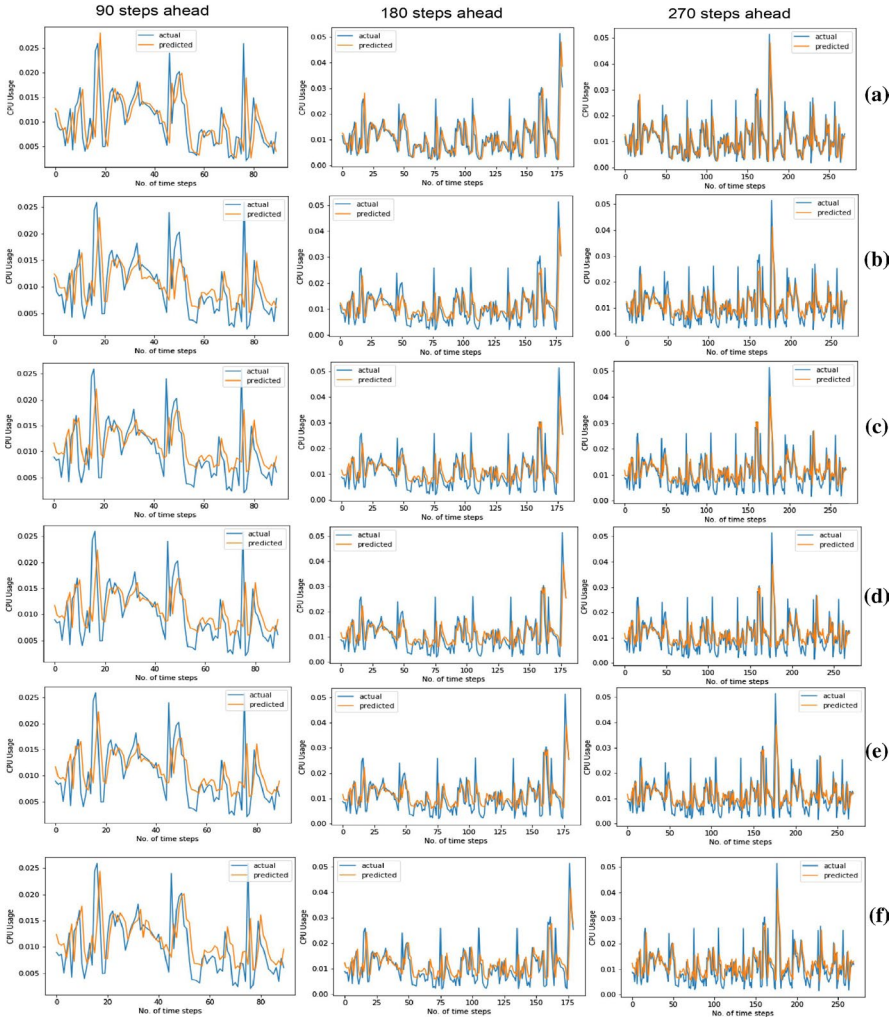


Fig. 16 Multi-steps-ahead CPU prediction with Pearson using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

to an explicit selection of VMs to mitigate hotspots and coldspots and prevent a VM's migration to a PM might result in the creation of new hotspots in the near future. Furthermore, the EQV-based VM selection increases energy savings and reduces the migration cost. The EQV metric always tries to assign VMs on PMs with less CPU utilization and smaller memory. Thus, it balances both the energy cost and migration cost.

6. *ESV* Figure 23b shows the performance comparison of ESV (product of Energy and SLAV) metric. The proposed scheme depicts significant improvement because it minimizes energy consumption and the SLAV violation rate. Thus, it maintains a trade-off between power cost and QoS guarantee.

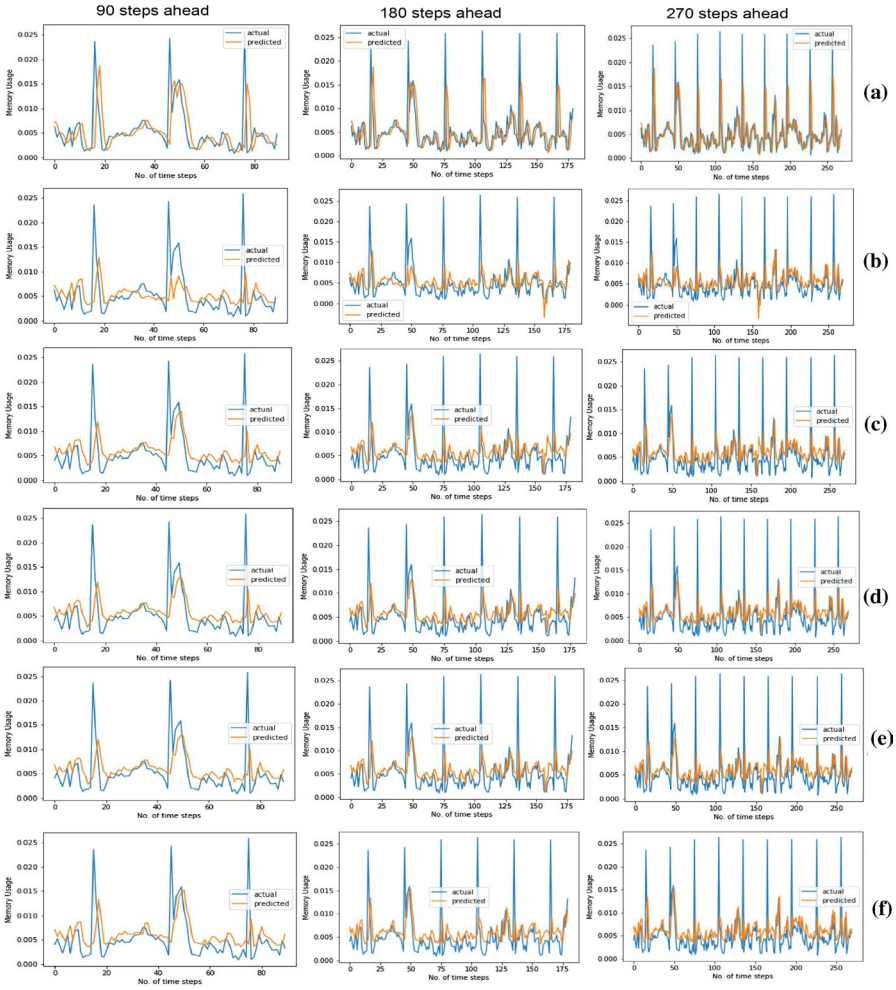


Fig. 17 Multi-steps-ahead memory prediction with Pearson using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

The results demonstrate that the proposed scheme remarkably outperforms the benchmarks in terms of prediction accuracy, energy consumption, migrations, SLAV, and ESV values.

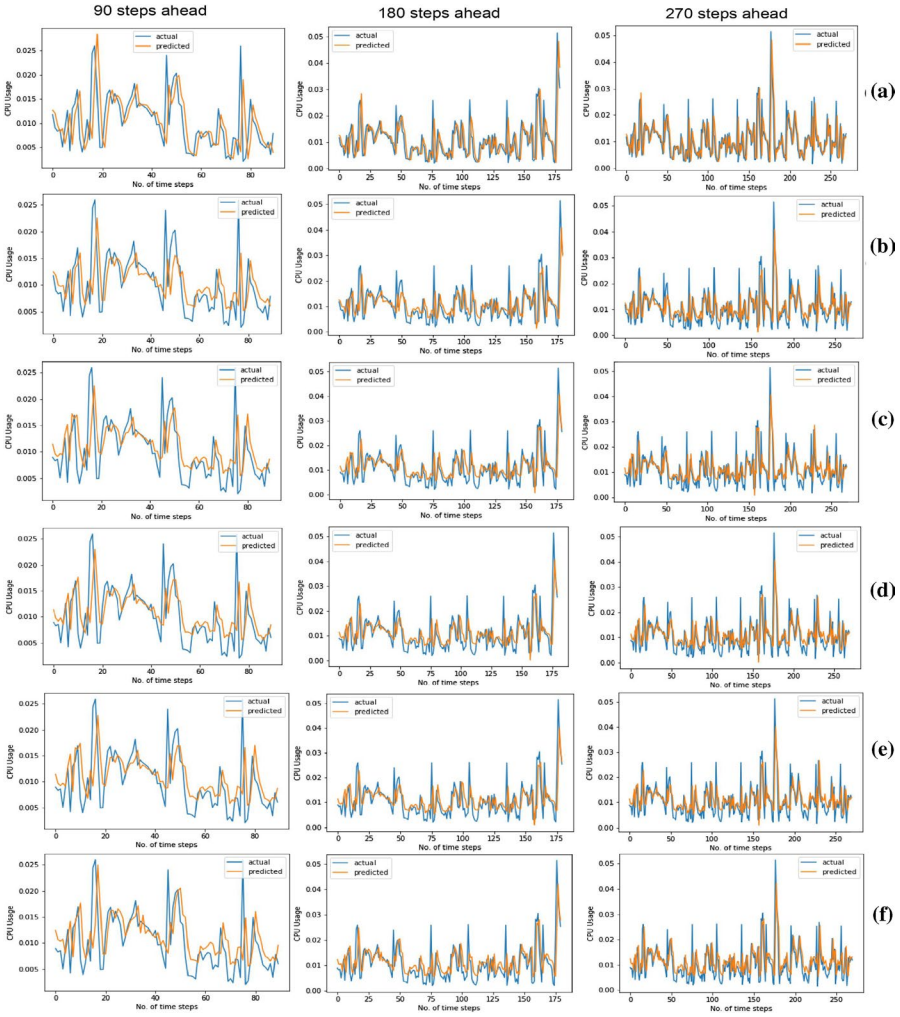


Fig. 18 Multi-steps-ahead CPU prediction with Spearman using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

7 Conclusions

This study presents deep learning-based multivariate resource utilization prediction for hotspots and coldspots mitigation in energy-efficient cloud data centers. We proposed a clustering-based stacked bidirectional LSTM deep learning network. We have divided our mitigation algorithm into different phases for hotspots and coldspots mitigation into various stages to generate an efficient mitigation plan. Firstly, it performs CPU and memory usage predictions using a deep learning model. Then, it considers both future and current resource usage to detect hotspots and coldspots formed in heterogeneous PMs. After their detection, we perform VM selection and

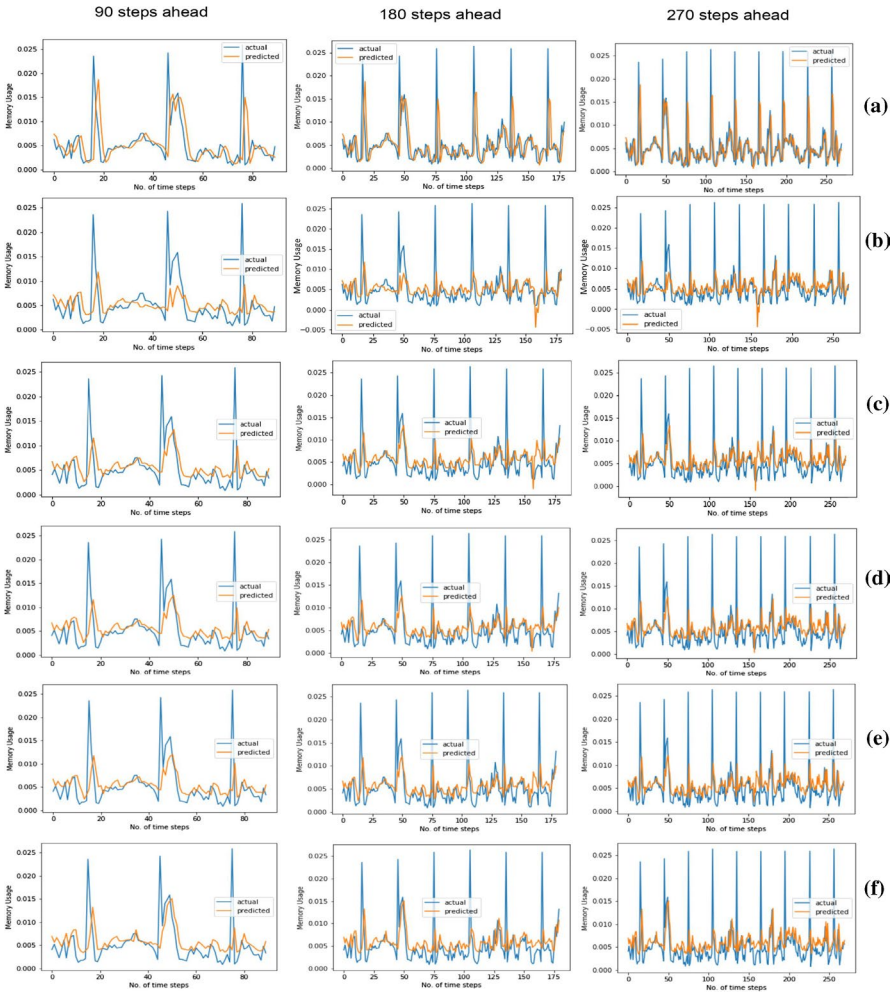


Fig. 19 Multi-steps-ahead memory prediction with Spearman using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

select the target PMs for VMs. Moreover, this study also presents trace-driven simulations using Google cluster workload traces. The obtained results show significant improvements over existing approaches concerning RMSE of prediction, energy-efficiency, migration cost, the number of actively used PMs, SLA violations, and ESV.

Future research might include considering network topology, communication cost, and implementation on the real-world cloud platform, such as OpenStack. Moreover, we have also observed some of the limitations of the proposed technique.

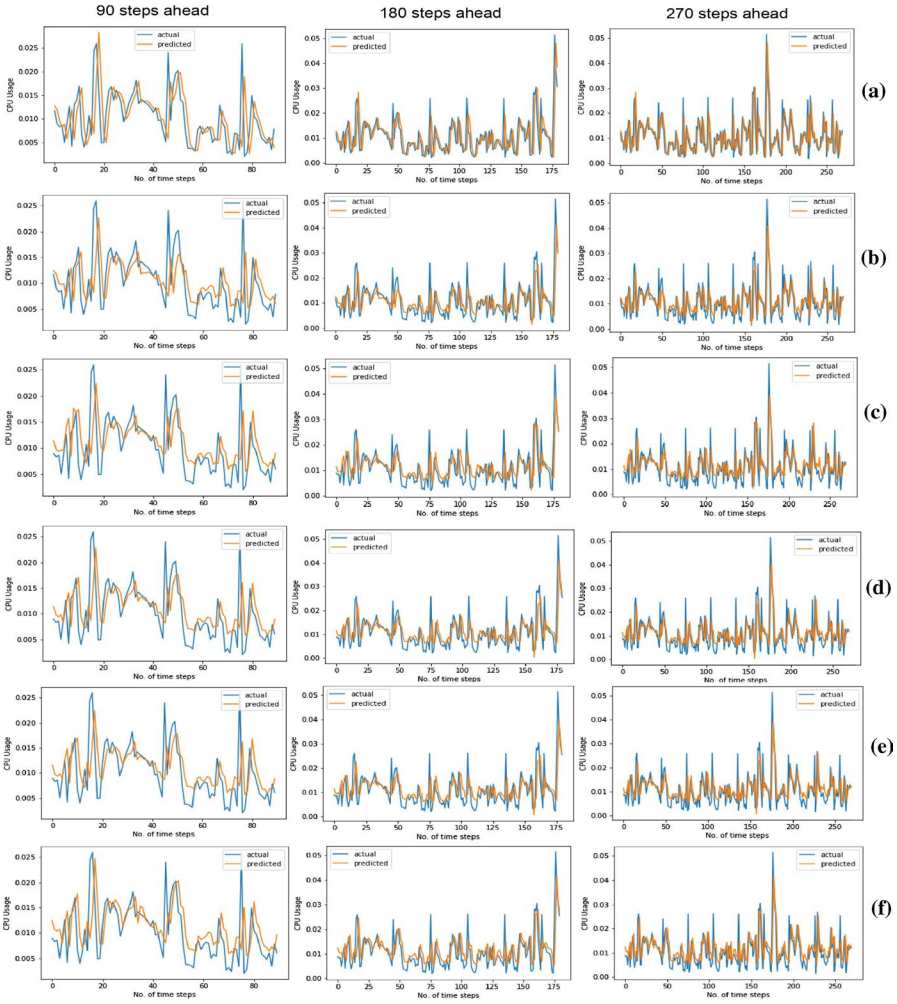


Fig. 20 Multi-steps-ahead CPU prediction with Kendall using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM

For instance, the deep learning models are suitable for small- and medium-scale cloud data enters, but for large-scale data centers, global machine learning models are best suited for overall resource usage prediction of VMs. Also, the consideration of concurrent migrations and resource over-subscription could result in different variations in migration and energy cost analysis.

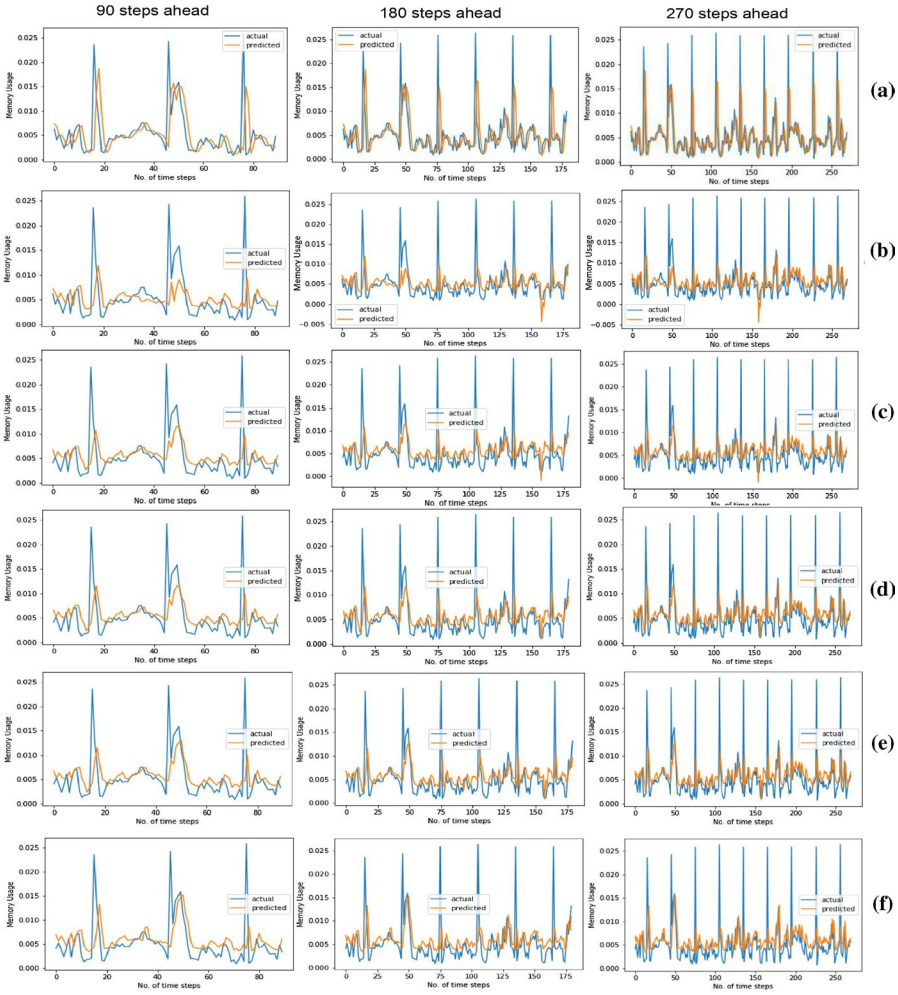
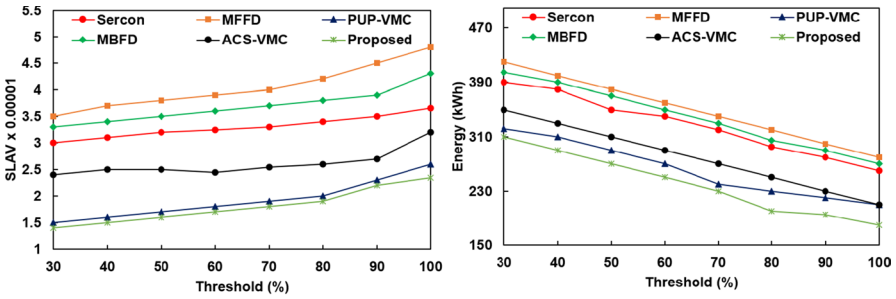
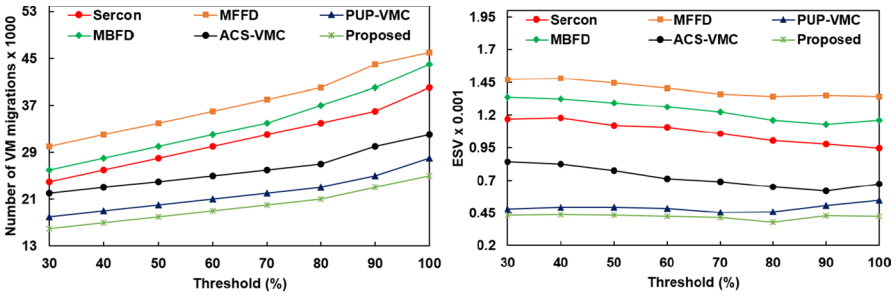


Fig. 21 Multi-steps-ahead memory prediction with Kendall using **a** ARIMA, **b** Linear Regression, **c** GRU, **d** LSTM, **e** BiLSTM, **f** Clustering-based Stacked BiLSTM



(a) Comparison of the SLAV metric **(b)** Comparison of the energy consumption

Fig. 22 Comparison of **a** The SLAV metric **b** Energy consumption for GCD workload trace



(a) Migration comparison **(b)** ESV metric comparison

Fig. 23 Analysis of **a** VM migrations, and **b** ESV metric

References

1. Buyya R, Yeo CS, 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 Gener Comput Syst 25(6):599–616
2. VMwareInc (2009) How VMware virtualization right sizes IT infrastructure to reduce power consumption. VMware Inc, Palo Alto, CA
3. Paul B, Boris D, Keir F, Steven H, Tim H, Alex H, Rolf N, Ian P, Andrew W (2003) Xen and the art of virtualization. SIGOPS Oper Syst Rev 37(5):164–177
4. Beloglazov A, Buyya R, Lee YC, Zomaya A (2011) A taxonomy and survey of energy-efficient data centers and cloud computing systems. Adv. Comput. 82:47–111
5. Asad Z, Chaudhry MAR (2017) A two-way street: green big data processing for a greener smart grid. IEEE Syst. J. 11(2):784–795
6. Gartner Inc (2007) Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions. Gartner Press Release
7. Global warming: Data centres to consume three times as much energy in next decade, experts warn (2016) https://www.independent.co.uk/environment/global-warming-data-centres-consume-three-times-much-energy-next-decade-experts-warn-a6830086.html. Accessed 24 Aug 2020
8. How to stop data centres from gobbling up the world's electricity (2018) https://www.nature.com/articles/d41586-018-06610-y. Accessed 27 Aug 2020
9. Barroso LA, Hölzle U (2007) The case for energy-proportional computing. Computer 12:33–37

10. Fan X, Weber WD, Barroso LA (2007) Power provisioning for a warehouse-sized computer. ACM SIGARCH Comput Archit News ACM 35:13–23
11. Christopher C (2005) Live migration of virtual machines. In: Proceedings of the 2nd conference on symposium on networked systems design & implementation. Berkeley, CA, USA, pp 273–286
12. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. Concurr Comput Pract Exp 24(13):1397–1420
13. Timothy W, Prashant S, Arun V, Mazin Y (2007) Black-box and gray-box strategies for virtual machine migration. In: Proc. 4th USENIX conference on networked systems design & implementation, NSDI'07, Cambridge, MA, USENIX Association, Berkeley, CA, USA, pp 229–242
14. Bin-packing (2006) In: Proc. combinatorial optimization, ser. Algorithms and combinatorics, vol 21. Springer Berlin Heidelberg, pp 426-441
15. Khan MA, Paplinski A, Khan AM, Murshed M, Buyya R (2018) Dynamic virtual machine consolidation algorithms for energy-efficient cloud resource management: a review. Sustainable cloud and energy services. Springer, New York, pp 135–165
16. Zhou Q, Xu M, Gill SS, Gao C, Tian W, Xu C, Buyya R (2020) Energy efficient algorithms based on VM consolidation for cloud computing: comparisons and evaluations. In: proc. 20th IEEE/ACM international symposium on cluster, cloud and internet computing (CCGRID). Melbourne, Australia, pp 489–498
17. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. 41(1):23–50
18. Farahnakian F, Liljeberg P, Plosila J (2013) LiRCUP: linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers. In: Proc. 39th euromicro conference on software engineering and advanced applications. Santander, pp 357–364
19. Farahnakian F, Pahikkala T, Liljeberg P, Plosila J (2013) Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data centers. In: Proc. IEEE/ACM 6th international conference on utility and cloud computing. Dresden, pp 256–259
20. Farahnakian F, Ashraf A, Pahikkala T, Liljeberg P, Plosila J, Porres I, Tenhunen H (2015) Using ant colony system to consolidate VMs for green cloud computing. IEEE Trans Ser Comput 8(2):187–198
21. Farahnakian F, Pahikkala T, Liljeberg P, Plosila J, Hieu NT, Tenhunen H (2019) Energy-aware VM consolidation in cloud data centers using utilization prediction model. IEEE Trans Cloud Comput 7(2):524–536
22. Moghaddam SM, O'Sullivan M, Walker C, Piraghaj SF, Unsworth CP (2020) Embedding individualized machine learning prediction models for energy efficient VM consolidation within Cloud data centers. Future Gener Comput Syst 106:221–233
23. Haghshenas K, Mohammadi S (2020) Prediction-based underutilized and destination host selection approaches for energy-efficient dynamic VM consolidation in data centers. J Supercomput 76:10240–10257
24. Hsieh S-Y, Liu C-S, Buyya R, Zomaya AY (2020) Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers. J Parallel Distrib Comput 139:99–109
25. Tarafdar A, Debnath M, Khatua S et al (2020) Energy and quality of service-aware virtual machine consolidation in a cloud data center. J Supercomput. 76:9095–9126
26. Lu S, Chen J (2018) Host overloading detection based on EWMA algorithm in cloud computing environment. In: Proc. IEEE 15th international conference on e-business engineering (ICEBE). Xi'An, China, pp 274–279
27. Donnell NM, Howley E, Duggan J (2020) Dynamic virtual machine consolidation using a multi-agent system to optimise energy efficiency in cloud computing. Future Gener Comput Syst 108:288–301
28. Monshizadeh Naeen H, Zeinali E, Toroghi Haghighat A (2020) Adaptive Markov-based approach for dynamic virtual machine consolidation in cloud data centers with quality-of-service constraints. Softw Pract Exp 50:161–183
29. Li Z, Xinrong Y, Lei Y, Guo S, Chang V (2020) Energy-efficient and quality-aware VM consolidation method. Future Gener Comput Syst 102:789–809
30. El-Moursy A, Abdelsamea A, Kamran R, Saad M (2019) Multi-dimensional regression host utilization algorithm (MDRHU) for host overload detection in cloud computing. J Cloud Comput 8:1–17

31. Ranjbari M, Torkestani JA (2018) A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers. J Parallel Distrib Comput 113:55–62
32. Subramanian S, Kannammal A (2019) Real time non-linear cloud workload forecasting using the holt-winter model. In: Proc. 10th international conference on computing, communication and networking technologies (ICCCNT). Kanpur, India, pp 1–6
33. Qi Z, Mohamed FZ, Shuo Z, Quanyan Z, Raouf B, Joseph LH (2012) Dynamic energy-aware capacity provisioning for cloud computing environments. In: Proc. international conference on autonomic computing (ICAC). ACM, New York, NY, USA, pp 145–154
34. Podolskiy V, Jindal A, Gerndt M, Oleynik Y (2018) Forecasting models for self-adaptive cloud applications: a comparative study. In: Proc. IEEE 12th international conference on self-adaptive and self-organizing systems (SASO). Trento, Italy, pp 40–49
35. Caglar F, Gokhale A (2014) iOverbook: intelligent resource overbooking to support soft real-time applications in the cloud. In: Proc. IEEE 7th international conference on cloud computing (CLOUD). IEEE, Anchorage, AK, USA, pp 538–545
36. Gong Z, Gu X, Wilkes J (2010) PRESS: PRedictive elastic resource scaling for cloud systems. In: Proc. international conference on network and service management (CNSM). IEEE, Niagara Falls, ON, Canada, pp 9–16
37. Nguyen H, Shen Z, Gu X, Subbiah S, Wilkes J (2013) AGILE: elastic distributed resource scaling for infrastructure-as-a-service. In: Proc. 10th international conference on autonomic computing (ICAC). San Jose, CA, USENIX, pp 69–82
38. Ghorbani M, Wang Y, Xue Y, Pedram M, Bogdan P (2014) Prediction and control of bursty cloud workloads: a fractal framework. In: Proc. international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 1–9
39. Reiss C, Wilkes J, Hellerstein JL (2011) Google cluster-usage traces: format + schema. https://github.com/google/cluster-data
40. Song B, Yu Y, Zhou Y et al (2018) Host load prediction with long short-term memory in cloud computing. J Supercomput. 74:6554–6568
41. Chakraborty K, Mehrotra K, Mohan CK, Ranka S (1992) Forecasting the behavior of multivariate time series using neural networks. Neural Netw. 5(6):961–970
42. Aboagye-Sarfo P et al (2015) A comparison of multivariate and univariate time series approaches to modelling and forecasting emergency department demand in Western Australia. J Biomed Inf 57:62–73
43. Dannecker L (2015) Energy time series forecasting: efficient and accurate forecasting of evolving time series from the energy domain, 1st edn. Springer, Berlin
44. Sun Y, Li J, Liu J, Chow C, Sun B, Wang R (2015) Using causal discovery for feature selection in multivariate numerical time series. Mach Learn 101(1–3):377–395
45. Zhao Y, Ye L, Li Z, Song X, Lang Y, Su J (2016) A novel bidirectional mechanism based on time series model for wind power forecasting. Appl Energy 177:793–803
46. Wakuya H, Shida K (2001) Bi-directionalization of neural computing architecture for time series prediction. III. Application to laser intensity time record Data Set A. In: Proc. international joint conference on neural networks (IJCNN). IEEE, Washington, DC, USA, pp 2098–2103
47. Cui Z, Ke R, Pu Z, Wang Y (2020) Stacked bidirectional and unidirectional LSTM recurrent neural network for forecasting network-wide traffic state with missing values. Transp Res Part C Emerg Technol 118:102674
48. Gupta S, Dinesh DA (2017) Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks. In: Proc. international conference on advanced networks and telecommunications systems (ANTS). IEEE, Bhubaneswar, pp 1–6
49. Gupta S, Dileep AD, Gonsalves TA (2018) A joint feature selection framework for multivariate resource usage prediction in cloud servers using stability and prediction performance. J Supercomput 74(11):6033–6068
50. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 28(5):755–768
51. Standard Performance Evaluation Corporation (SPEC) (2008), http://www.spec.org
52. Blackburn M, Grid G (2008) Five ways to reduce data center server power consumption (white paper). The Green Grid
53. Boutaba R, Zhang Q, Zhani MF (2014) Virtual machine migration in cloud computing environments: benefits, challenges, and approaches. Communication infrastructures for cloud computing. IGI Global, Pennsylvania, pp 383–408

54. Zakarya M, Gillam L (2019) Managing energy, performance and cost in large scale heterogeneous datacenters using migrations. Future Gener Comput Syst. 93:529–547
55. Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45(11):2673–2681
56. Wang M, Meng X, Zhang L (2011) Consolidating virtual machines with dynamic bandwidth demand in data centers. In: 2011 Proceedings IEEE INFOCOM, vol 201. pp 71–75
57. Behera S, Misra R, Sillitti A (2021) Multiscale deep bidirectional gated recurrent neural networks based prognostic method for complex non-linear degradation systems. Inf Sci 554:120–144
58. Murtazaev Aziz, Sangyoon Oh (2011) Sercon: server consolidation algorithm using live migration of virtual machines for green computing. IETE Tech Rev 28(3):212–231

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Yashwant Singh Patel[1] · Rishabh Jaiswal[2] · Rajiv Misra[3]

Rishabh Jaiswal
rsj.rishabh@gmail.com

Rajiv Misra
rajivm@iitp.ac.in

[1] Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala, Punjab, India

[2] Department of Computer and Information Science and Engineering, University of Florida, 432 Newell Dr, Gainesville, FL 32611, USA

[3] Department of Computer Science and Engineering, Indian Institute of Technology Patna, Bihar, India

Server Virtualization

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss server virtualization and the need of routing and switching for physical and virtual machines.

- Then we will discuss the two methods of virtualization: **(i) Docker based and (ii) Linux container based** and also address the problem of networking VMs and its **hardware based approach: SR-IOV, single-root I/O virtualization and software based approach: Open vSwitch**

Cloud Computing depends on Server Virtualization

- **In the context of server virtualization, how do we network the large number of VMs that might reside on a single physical server?** Cloud computing depends heavily on server virtualization for several reasons:

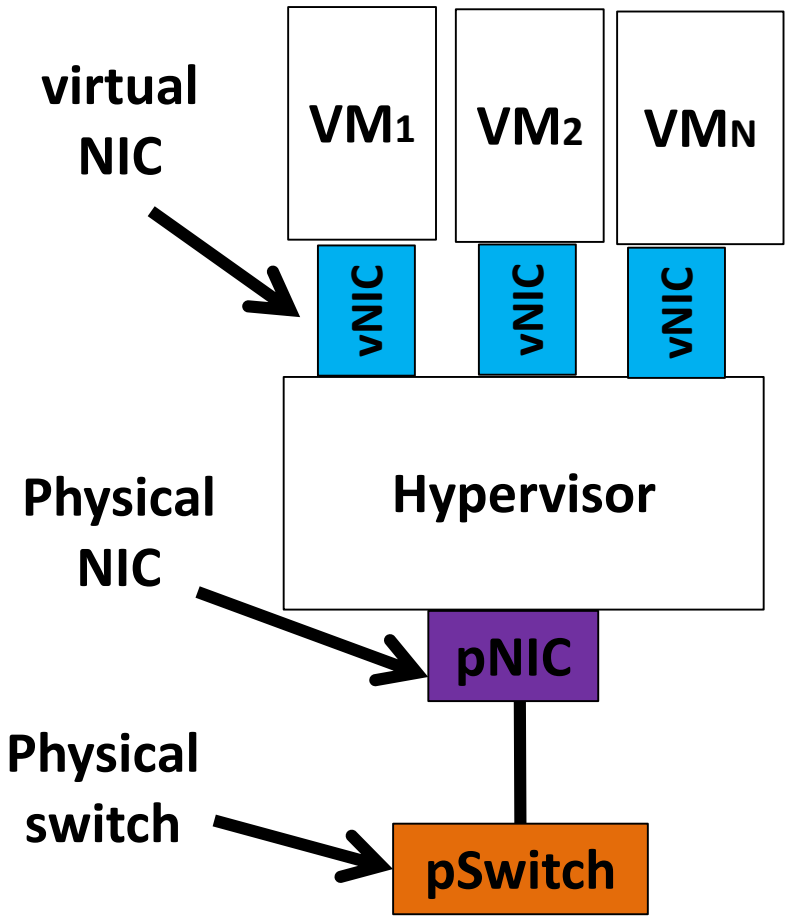
(i) **Sharing of physical infrastructure:** Virtual machines allow multiplexing of hardware with tens to 100s of VMs residing on the same physical server. Also, it allows rapid deployment of new services.

(ii) **Spinning up a virtual machine in seconds:** Spinning up a virtual machine might only need seconds compared to deploying an app on physical hardware, which can take much longer.

(ii) **Live VM migration:** Further, if a workload requires migration. For example, you do physical server requiring maintenance. This can be done quickly with virtual machines, which can be migrated to other servers without requiring interruption of service in many instances. Due to these advantages today, more endpoints on the network are virtual rather than physical.

Server Virtualization

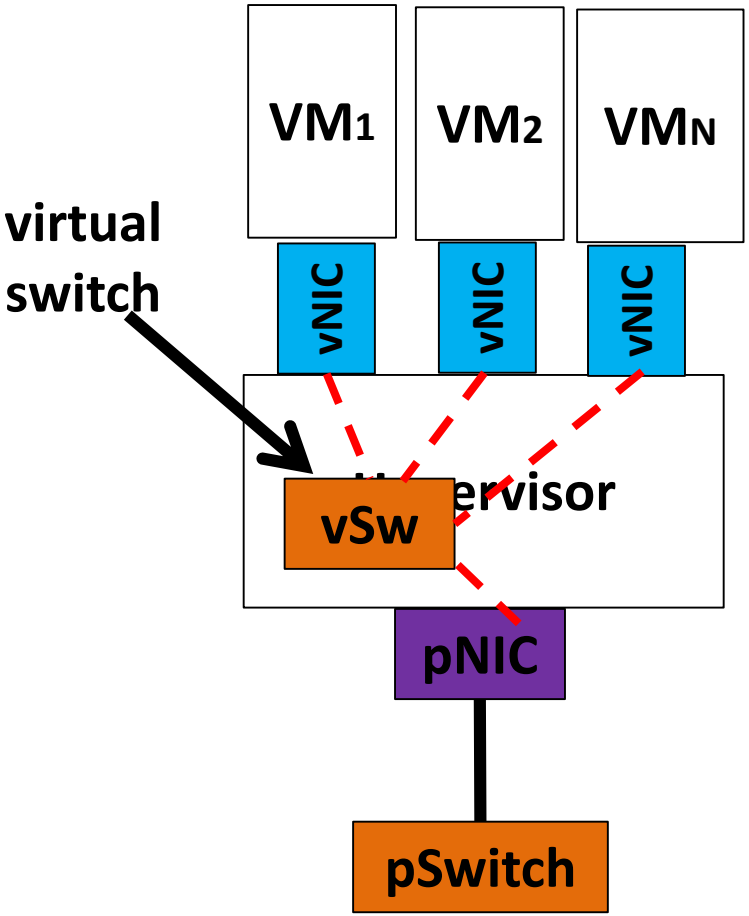
- The physical hardware is managed by a Hypervisor. This could be **Xen, KVM, or VMWare's ESXI,** or multiple such alternatives.
- On top of the Hypervisor runs several VMs in user space.
- The hypervisor provides an emulated view of the hardware to the VMs, which the VMs treat as their substrate to run a guest OS on.
- **Among other hardware resources the network interface card is also virtualized in this manner.** The hypervisor managing the physical NIC, is exposing virtual network interfaces to the VMs. **The physical NIC also connects the server to the rest of the network.**



Networking of VMs inside the Hypervisor

- **The hypervisor runs a virtual switch,** this can be a simple layer tool searching device, Operating in software inside the hypervisor.

- vSw is **connected to all the virtual NICs,** has them as the physical NIC, and moved packets between the VMs and the external network.



Alternate Methods of virtualization:

(i) Using Docker

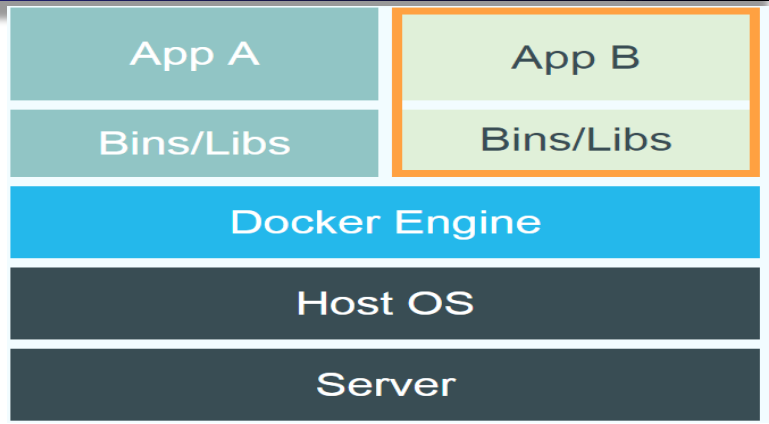
(ii) Using Linux containers

Docker

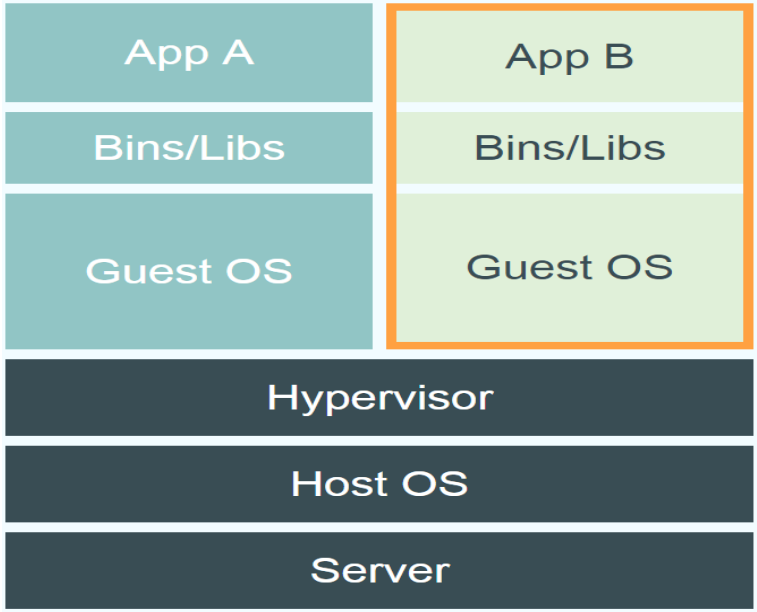
- **Docker is an open-source project that automates the deployment of applications inside software containers,** by providing an additional layer of abstraction and automation of OS–level virtualization on Linux.

- **The Docker Engine container comprises just the application and its dependencies.**

- **It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers.** Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.



Docker

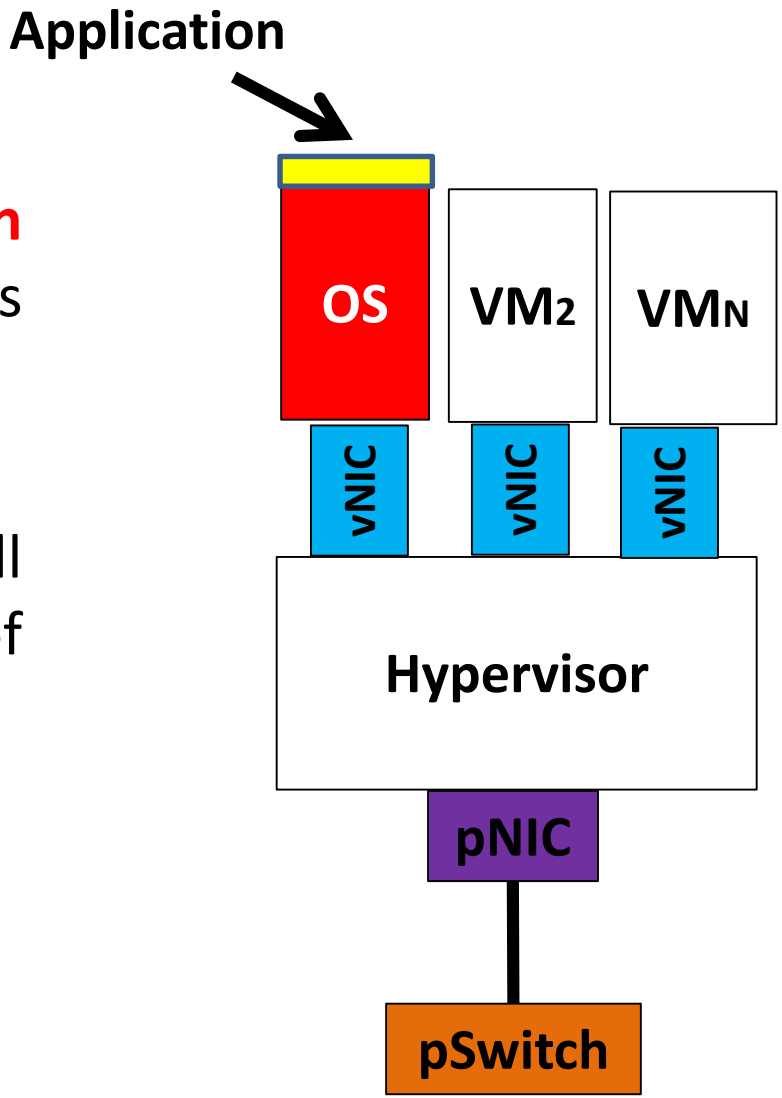


Virtual Machine

(i) Using VMs as Virtualization

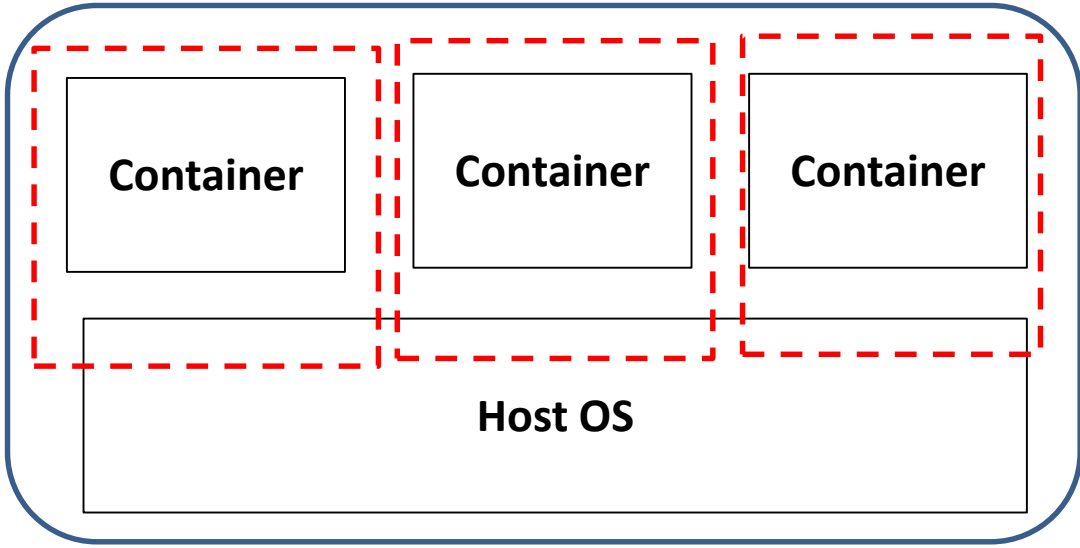
- In this model, **each VM runs its own entire guest OS.** The application runs as a process inside this guest OS.

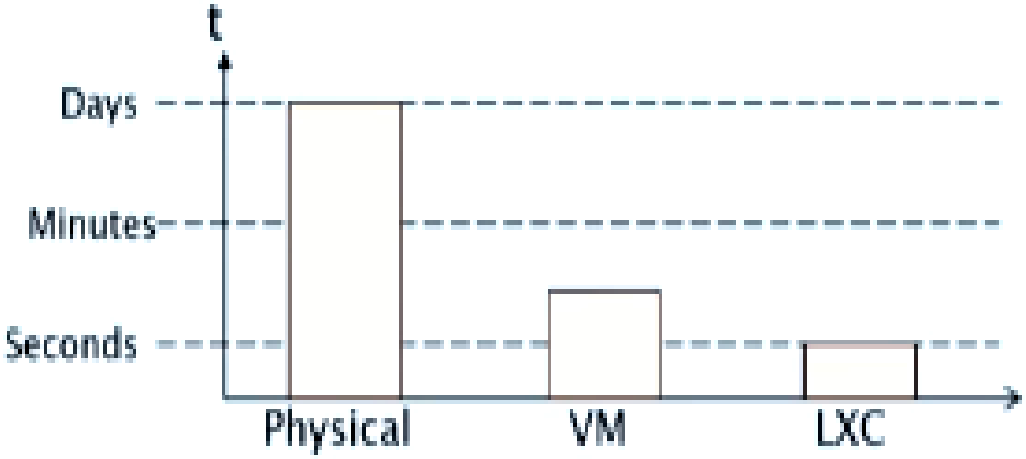
- This means that even running a small application requires the overhead of running an entire guest OS.



Linux containers as Virtualization

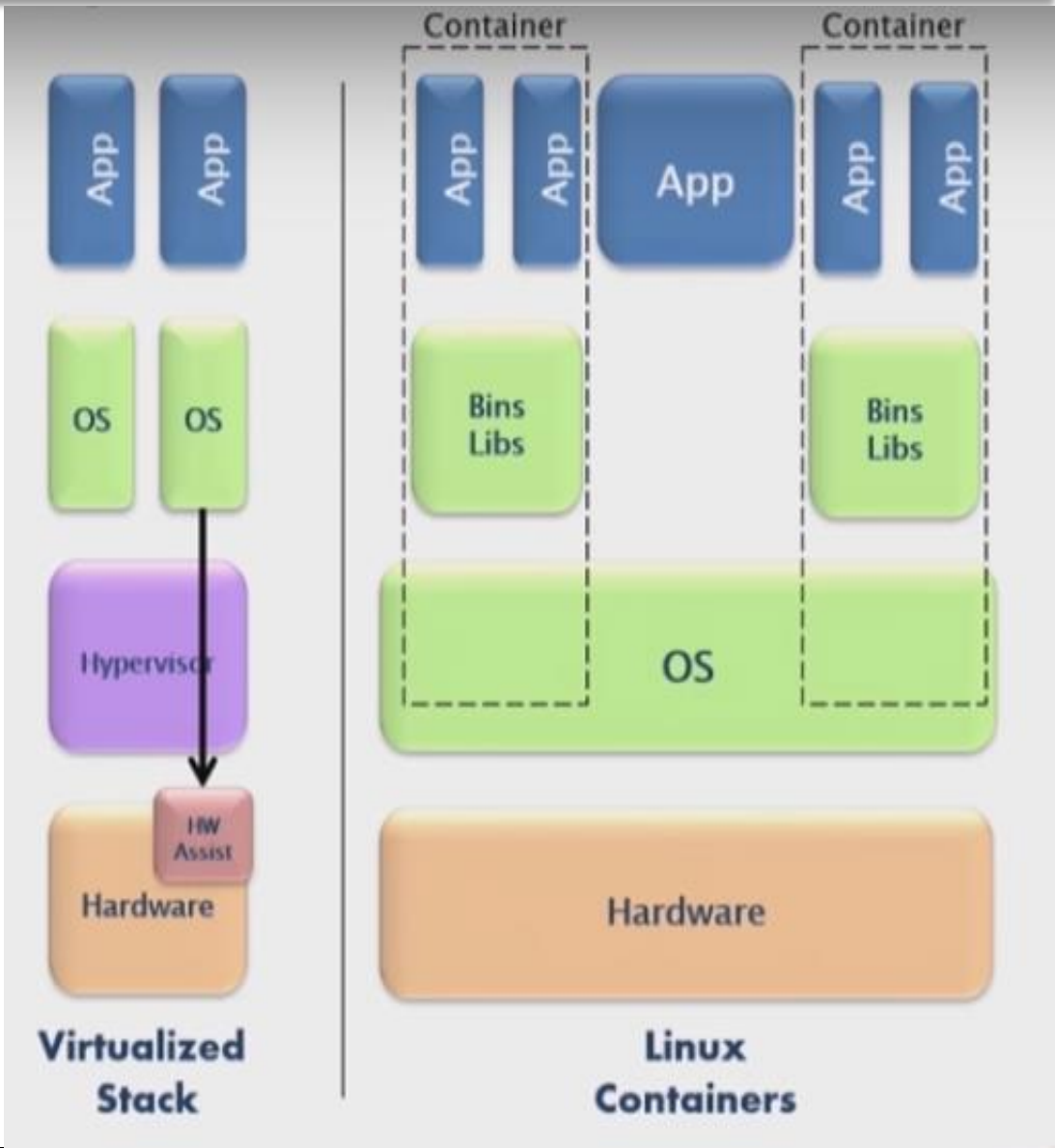
- Create an environment as close as possible to a standard Linux without separate kernel
- **Uses kernel features for separation**
- **Near bare metal (physical hardware) performance**
- **Fast provisioning times (Building up/start-up time)**





Linux containers

- Containers run in host systems kernel
- Separated with policies
- Can use apps in host system
- Can install additional libraries and apps
- Portable between OS variants supporting linux container
- No overhead with hypervisor and guest OS kernel

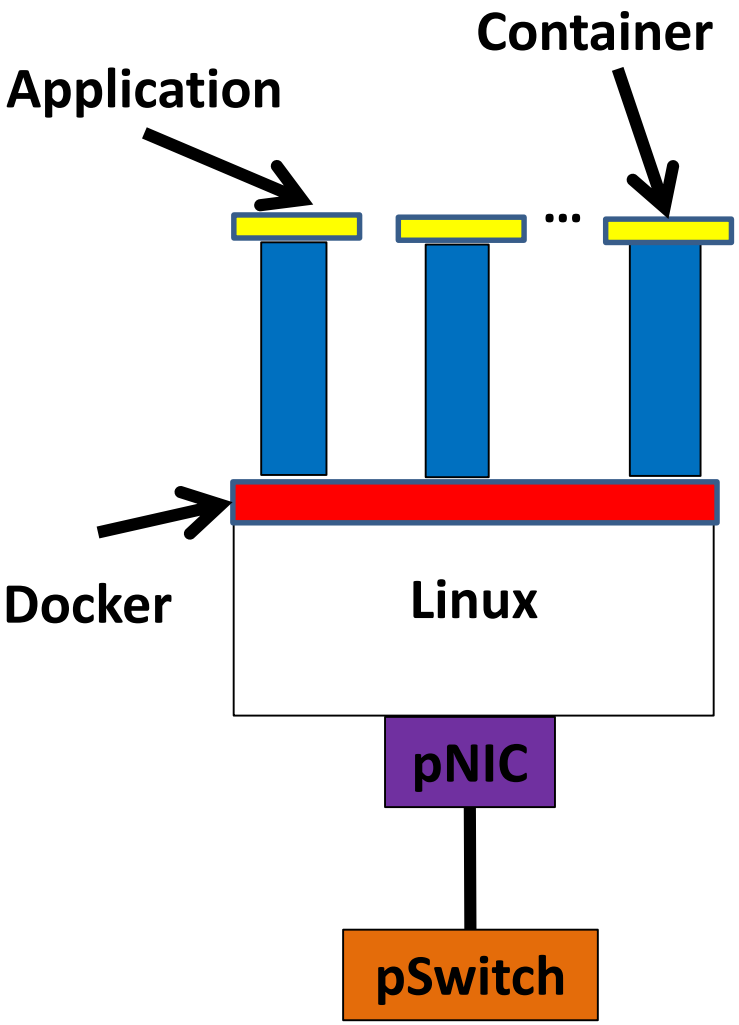


(ii) Using Linux containers

In this approach, an application together with its dependencies uses packages into a Linux container, which runs using the host machine's Linux stack and any shared resources.

Docker is simply a container manager for multiple such containers. Applications are isolated form each other by the use of separate namespaces. Resources in one application cannot be addressed by other applications. This yields isolation quite similar to VMs, but with a smaller footprint.

Further, containers can be brought up much faster than VMs. Hundreds of milliseconds as opposed to seconds or even tens of seconds with VMs.

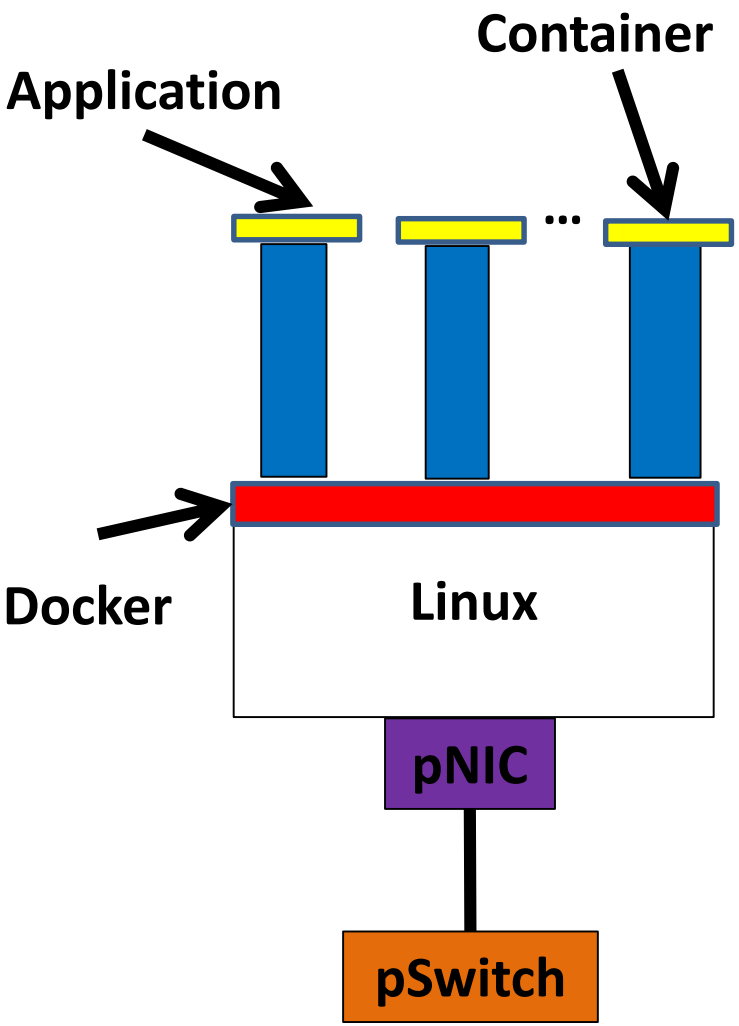


Networking with Docker

- **Each container is assigned a virtual interface.** Docker contains a virtual ethernet bridge connecting these multiple virtual interfaces and the physical NIC.

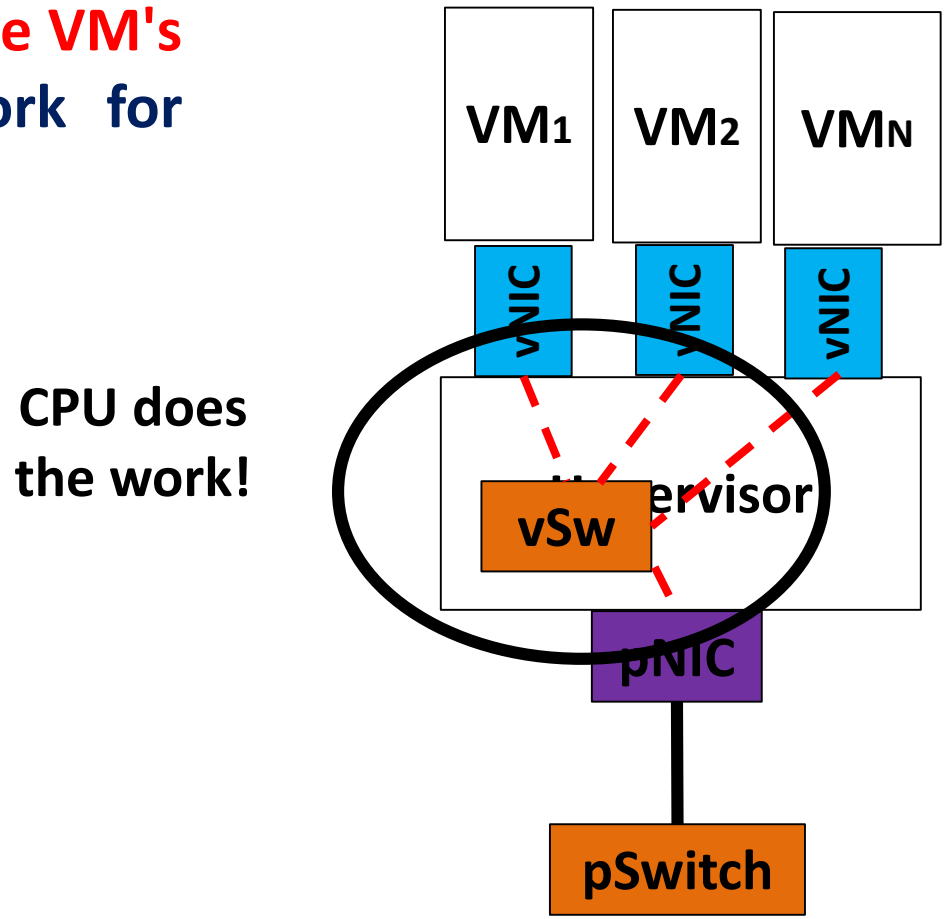
- **Configuring Docker and the environment variables decide what connectivity is provided.** Which machines can talk to each other, which machines can talk to the external network, and so on.

- **External network connectivity is provided through a NAT, that is a network address translator.**



Improving networking performance

- The **hypervisor runs a virtual switch to able to network the VM's** and **CPU is doing the work for moving the packets.**



Packet processing on CPUs

Flexible slow, CPU-expensive: Now packet processing on CPUs can be quite flexible because it can have general purpose forwarding logic. You can have packet filters on arbitrary fields run multiple packet filters if necessary, etc. But if done naively, this can also be **very CPU-expensive and slow.**

Packet forwarding: The packet forwarding entails: At 10Gbps line rates with the smallest packets, that's 84 Bytes. We only have an interval of 67ns before the next packet comes in on which we need to make a forwarding decision. Note that ethernet frames are 64 bytes, but together with the preamble which the tells the receiver that a packet is coming and the necessary gap between packets. The envelope becomes 84 bytes. For context a CPU to memory access takes tens of nanoseconds. So, 67 nanoseconds is really quite small.

Packet processing on CPUs

Need time for Packet I/O: Moving packets from the NIC buffers to the OS buffers, which requires CPU interrupts. Until recently a single X86 core couldn't even saturate a ten gigabits per second link. And this is without any switching required. This was just moving packets from the NIC to the OS. After significant engineering effort, packet I/O is now doable at those line rates. However for a software switch we need more.

Userspace overheads: If any of the switching logic is in userspace, one incurs the overhead for switching between userspace and kernel space.

Packet classification: Further, for switching we need to match rules for forwarding packets to a forwarding table. All of this takes CPU time. Also keep in mind that forwarding packets is not the main goal for the CPU. The CPU is there to be doing useful computation.

Approaches for Networking of VMs

There are two different approaches to address the problem of networking VMs:

(i) One, using specialized hardware:

- **SR-IOV, single-root I/O virtualization**

(ii) Other using an all software approach:

- **Open vSwitch**

(i) Hardware based approach

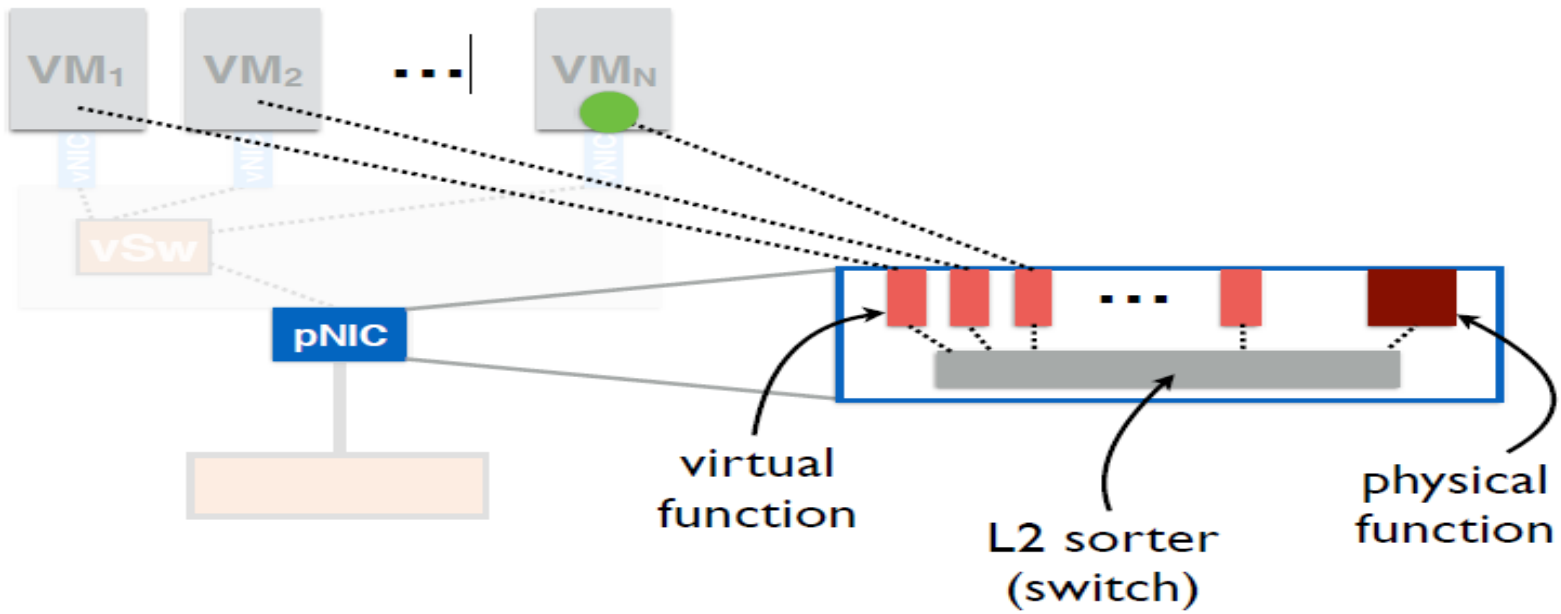
- **The main idea behind the hardware approach is that CPUs are not designed to forward packets, but the NIC is.**

- The naive solution would be to just give access to the VMs, to the NIC. But then problems arise. **How do you share the NIC's resources? How do you isolate various virtual machines?**

- **SR-IOV, single-root I/O virtualization**, based NIC provides one solution to this problem.

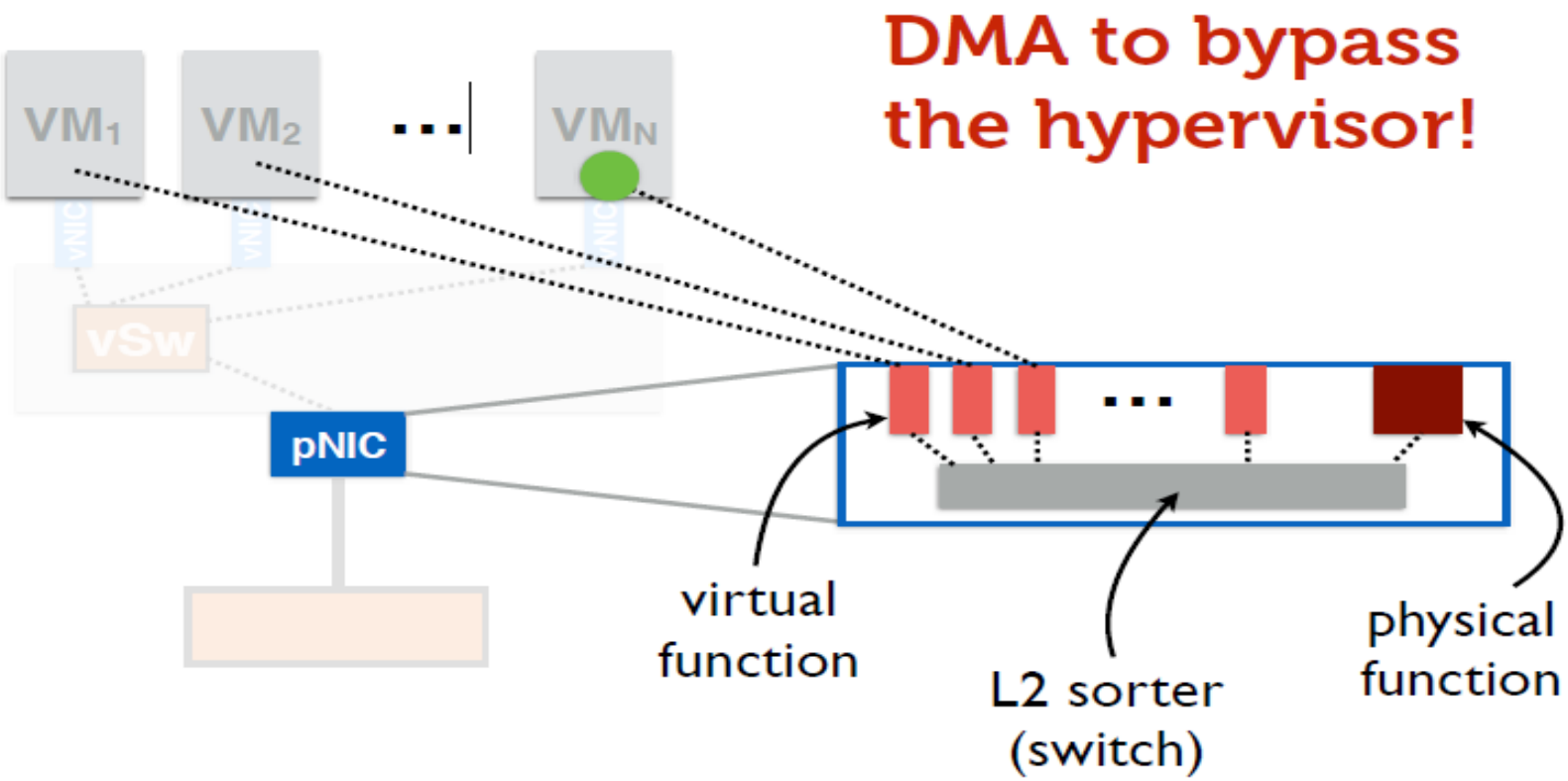
SR-IOV: Single-root I/O Virtualization

- **The SR-IOV, the physical link itself, supports virtualization in hardware.** So let's see inside this SR-IOV enabled network interface card.
- The NIC provides a physical function, which is just a standard ethernet port. In addition, it also provides several virtual functions which are simple queues that transmit and receive functionality.
- **Each VM is mapped to one of these virtual functions. So the VMs themselves get NIC hardware resources.**



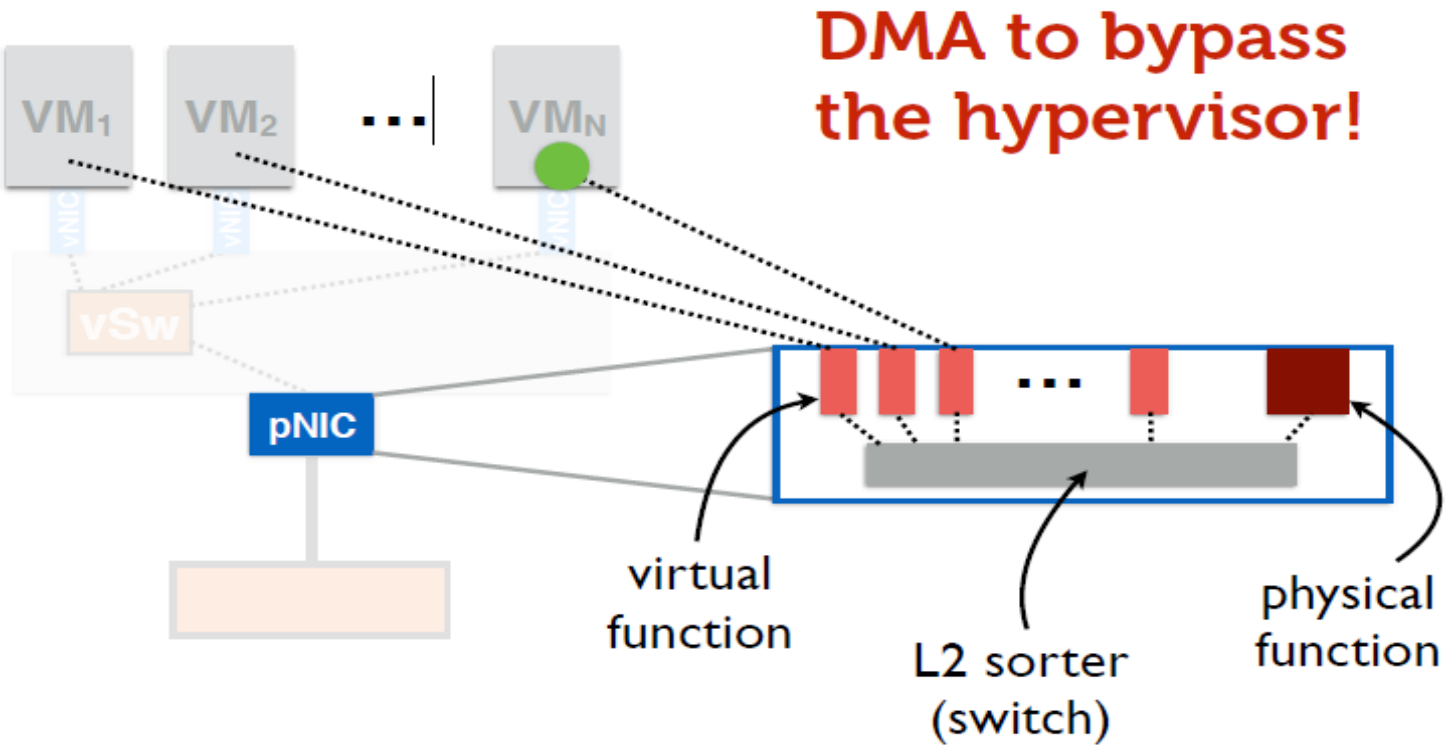
SR-IOV: Single-root I/O Virtualization

- **On the NIC there also resides a simple layer two, which classifies traffic into queues responding to these virtual functions**. Further, packets are moved directly from the net virtual function to the responding VM memory using **DMA (direct memory access).** This allows us to bypass the hypervisor entirely.



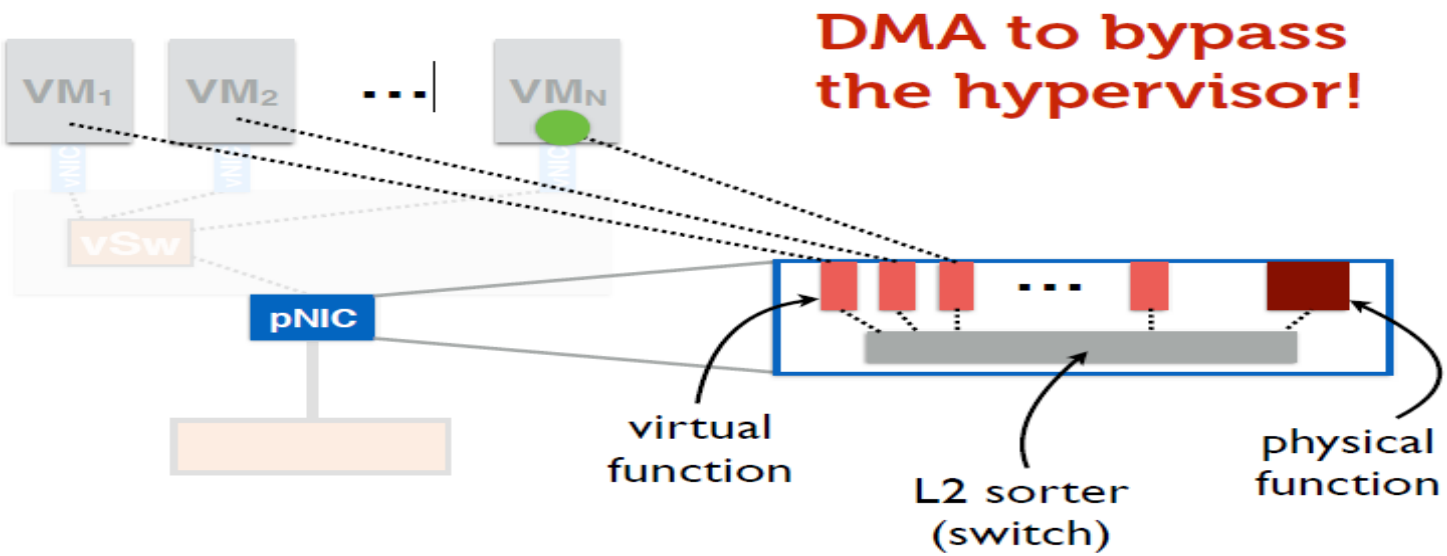
SR-IOV: Single-root I/O Virtualization

- **The hypervisor is only involved in the assignment of virtual functions to virtual machines.** And the management of the physical function, but not the data part for packets. The upshot to this is **higher through-put, lower latency and lower CPU utilization** and this give us close to native performance.



SR-IOV: Single-root I/O Virtualization

- There are downsides to this approach though. For one, **live VM migration becomes trickier**, because now you've tied the virtual machine to physical resources on that machine. The forwarding state for that virtual machine now resides in the layer two switch inside the NIC.
- **Second, forwarding is no longer as flexible.** We're relying on a layer two switch that is built into the hardware of the NIC. So we cannot have general purpose rules and we cannot be changing this logic very often. It's built into the hardware. **In contrast, software-defined networking (SDN) allows a much more flexible forwarding approach.**



(ii) Software based approach

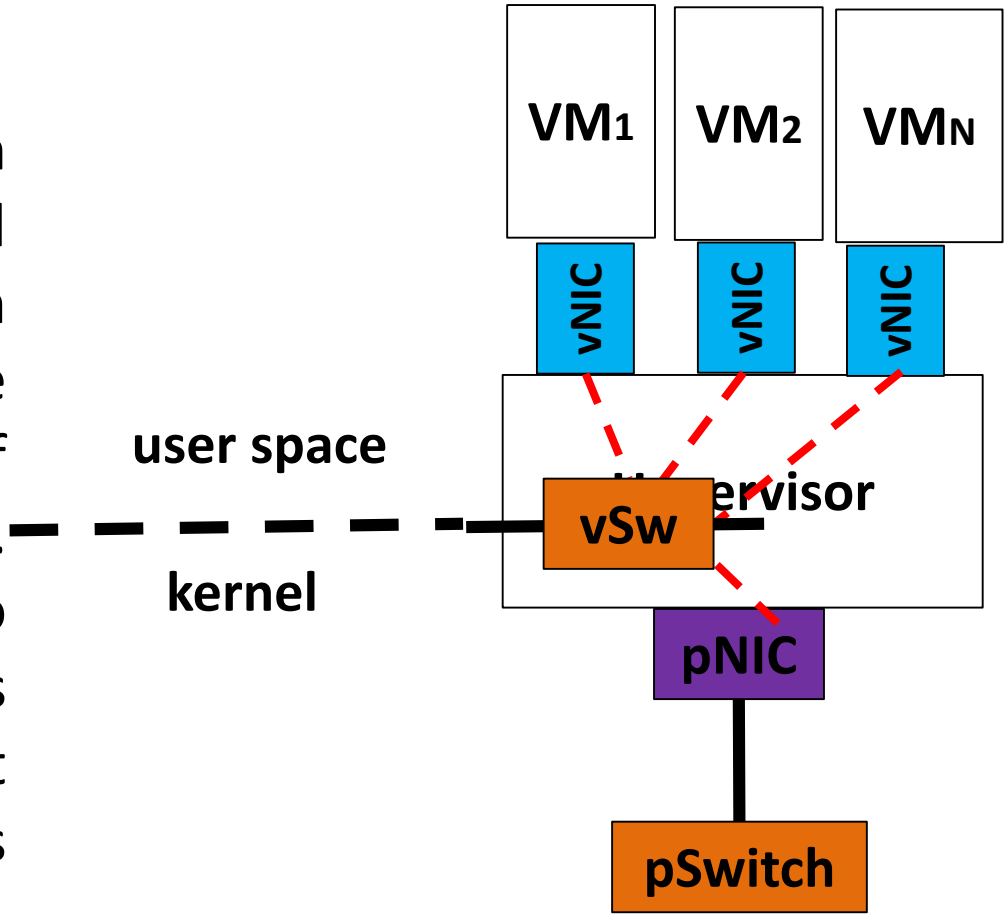
The software based approach that addresses:

(i) Much more flexible forwarding approach

(ii) Live VM migration

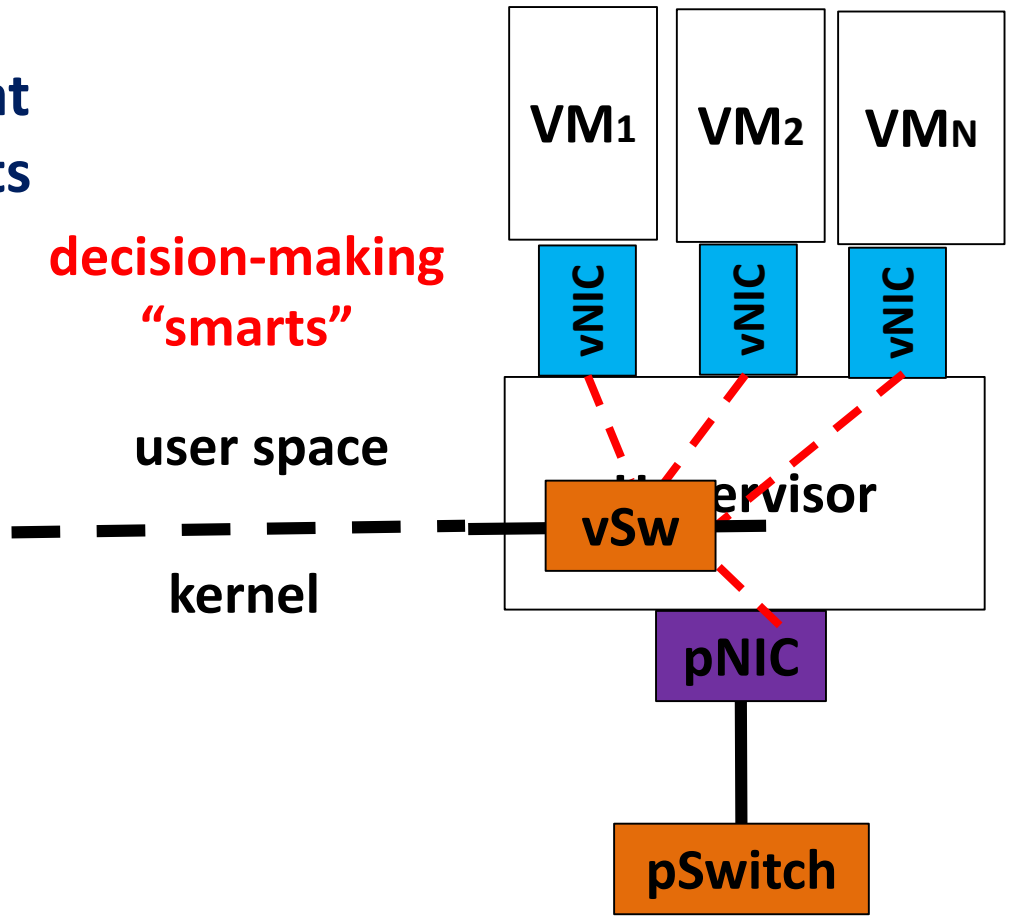
Open vSwitch

- Open vSwitch design goals are **flexible and fast-forwarding.**
- This necessitates a division between **user space** and **kernel space** task. One can not work entirely in the kernel, because of development difficulties. It's hard to push changes to kernel level code, and it's desirable to keep logic that resides in the kernel as simple as possible.



Open vSwitch

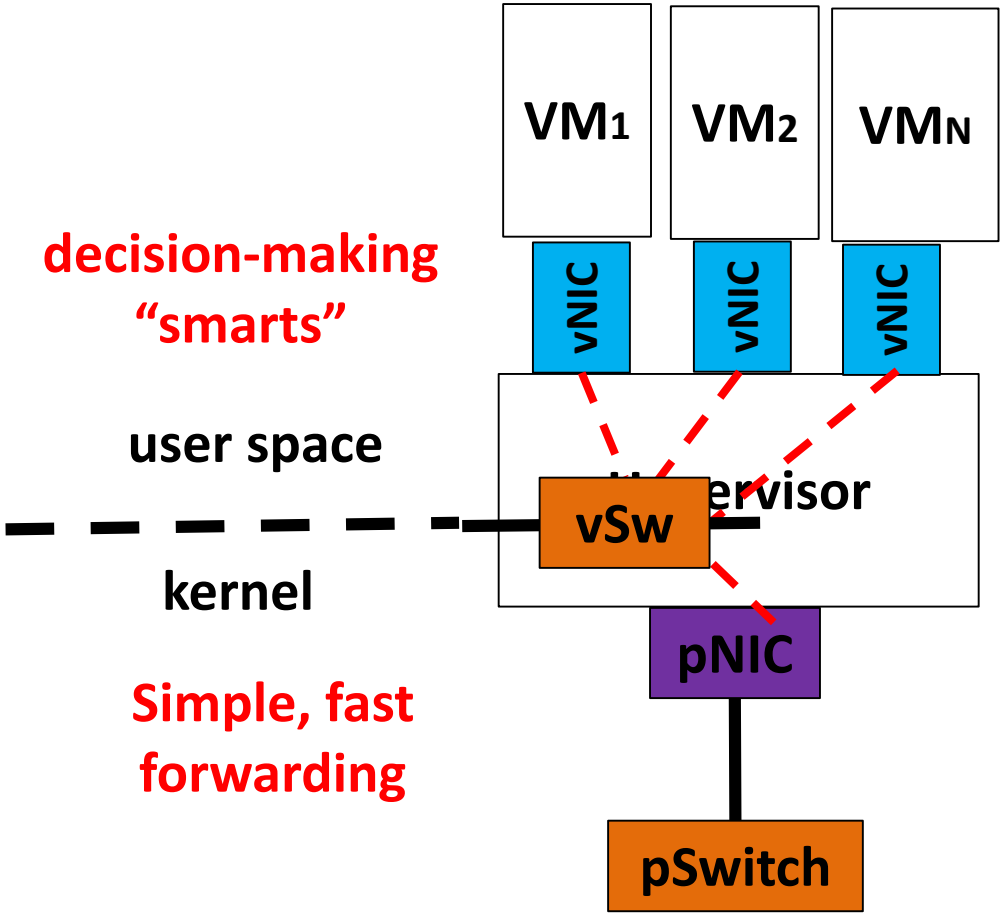
- So, the smarts of this approach, is the **switch routing decisions** lie in user space.
- This is where one **decides what rules or filters apply to packets of a certain type.** Perhaps based on network updates from other, possibly virtual, such as in the network. This behavior can also be programmed using **open flow.** **So, this part is optimized for processing network updates, and not necessarily for wire speed packet forwarding.**



Open vSwitch

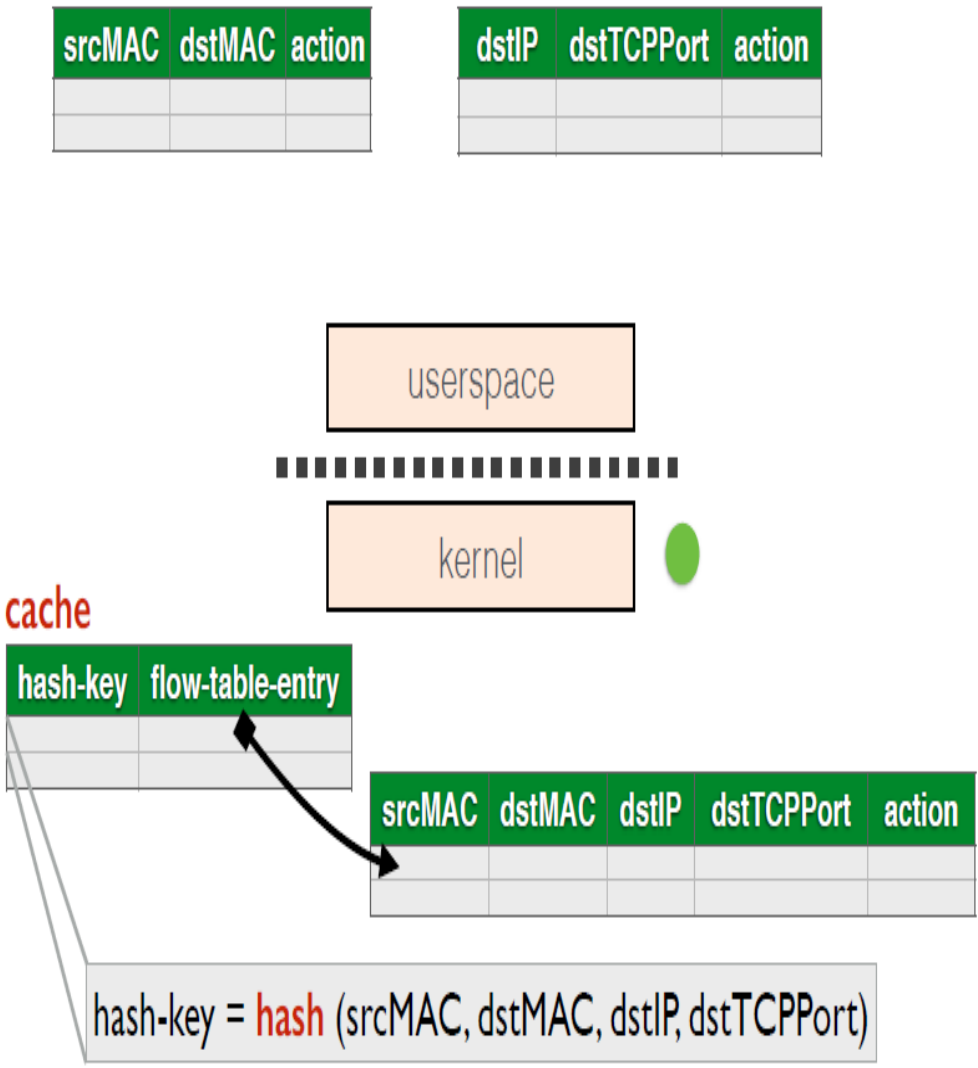
- **Packet forwarding**, on the other hand, is handled largely in the kernel, broadly.

- **Open vSwitch approach is to optimize the common case, as opposed to the worst case line rate requirements and caching will be the answer to that need.**



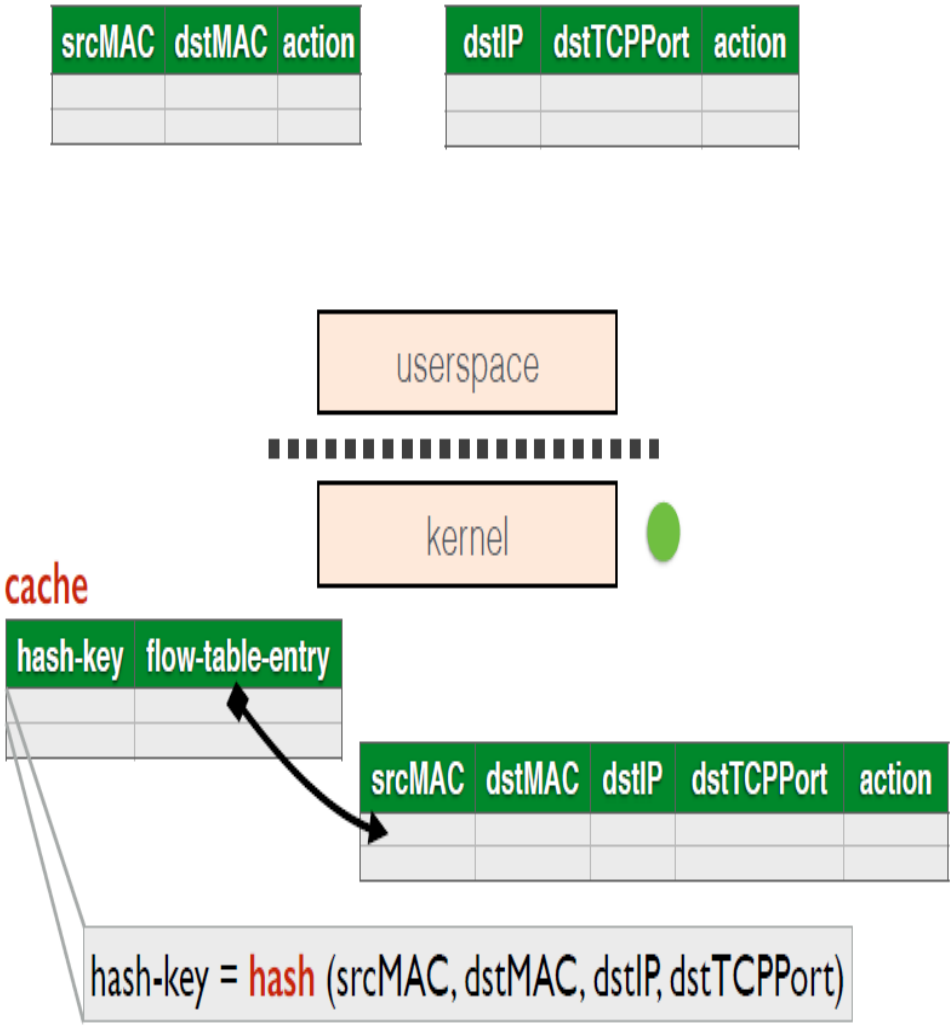
Inside Open vSwitch

- **The first packet of a flow goes to userspace here several different packet classifiers may be consulted.** Some actions may be based on MAC addresses and some others might depend on TCP PORTs, etc.
- The **highest priority matching action** across these different classifiers will be used to forward the packet.
- **Once a packet is forwarded, a collapsed rule used to forward that packet is installed in the kernel. This is a simple classifier with no priorities.** The following packets of this flow will never enter user space, seeing only the kernel level classifier.



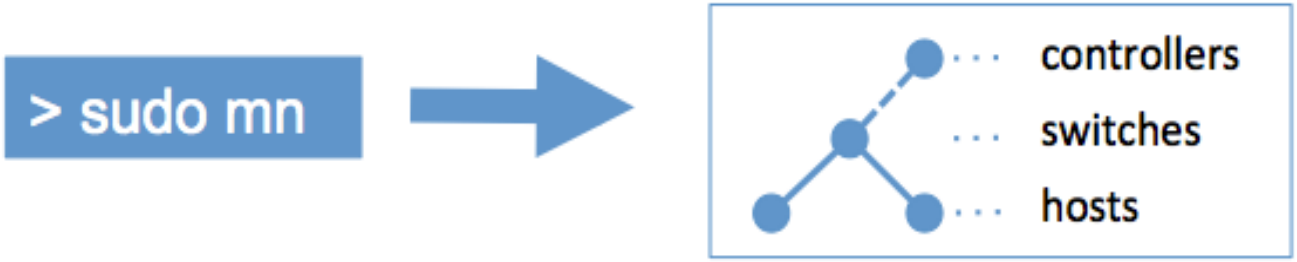
Inside Open vSwitch

- The problem though is when it's still running a packet classifier in the kernel in software. What this means is for **every packet that comes in, you are searching in this table for the right entry that matters and using that entry for forward the packet. This can be quite slow.**
- **Open vSwitch solves this problem is to create a simple hash table based cache into the classifier.** So instead of looking at this entire table and finding the right rule. You'll hash what fields are used to match packet, and the hash key is now your pointer to the action that needs to be taken. And, these hash keys and their actions can be cache.



Introduction to Mininet

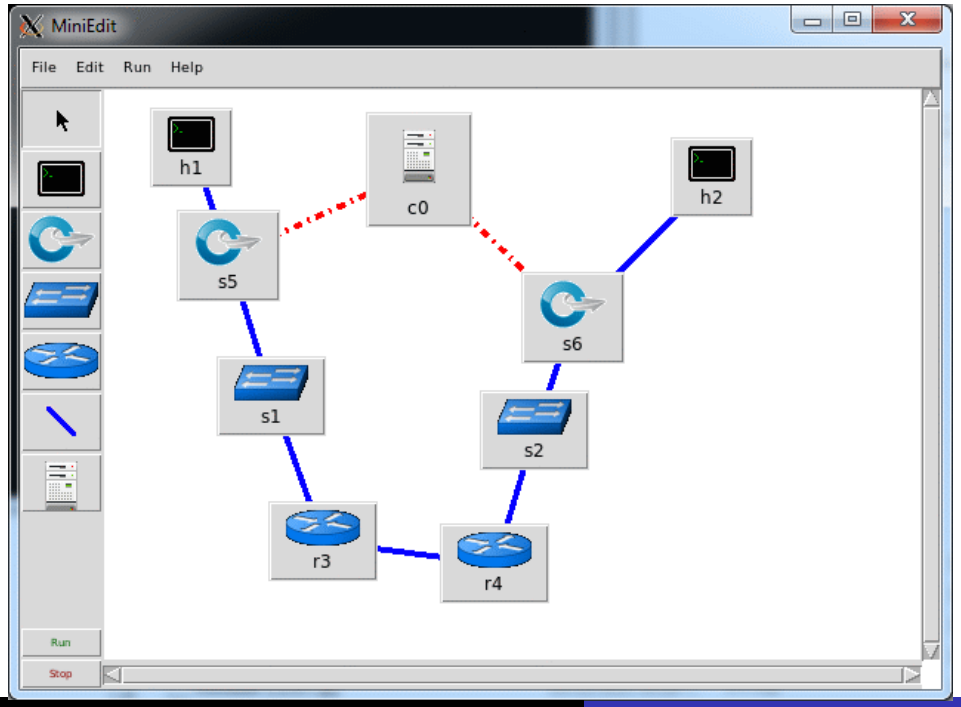
- Mininet creates a **realistic virtual network, running real kernel, switch and application code,** on a single machine (VM, cloud or native), in seconds, with a single command:



- It is a **network emulator** which creates realistic virtual network
  - Runs real kernel, switch and application code on a single machine
  - Provides both Command Line Interface (CLI) and Application Programming Interface (API)
    - **CLI: interactive commanding**
    - **API: automation**
  - **Abstraction**
    - Host: emulated as an OS level process
    - Switch: emulated by using software-based switch
      - **E.g., Open vSwitch, SoftSwitch**

Mininet Applications

- **MiniEdit (Mininet GUI)**
  - A GUI application which eases the Mininet topology generation
  - Either save the topology or export as a Mininet python script
- **Visual Network Description (VND)**
  - A GUI tool which allows automate creation of Mininet and OpenFlow controller scripts



Important Links for Mininet

- mininet.org
- github.com/mininet
- github.com/mininet/mininet/wiki/Documentation
- reproducingnetworkresearch.wordpress.com

Comparison

- The **hardware based approach** the **SR-IOV** takes **sacrifices flexibility and forwarding logic** for line rate performance in all scenarios. Virtually hitting native performance.

- While the **software based approach** **Open vSwitch**, the compromise made is to avoid targeting worst case performance, and **focusing on forwarding flexibility**.

References

- W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "**An updated performance comparison of virtual machines and Linux containers**," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, 2015, pp. 171-172.

- Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado, "**The design and implementation of open vSwitch**" In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15). USENIX Association, Berkeley, CA, USA, 2015, pp. 117-130.

Conclusion

- In this lecture, we have discussed **server virtualization and also discuss the need of routing and switching for physical and virtual machines.**

- We have discussed two methods of virtualization: **(i) Docker based and (ii) Linux container based.**

- To address the problem of networking VMs, two approaches are discussed: **(i) One, using specialized hardware: SR-IOV, single-root I/O virtualization and (ii) Other using an all software approach: Open vSwitch**

Virtualization

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

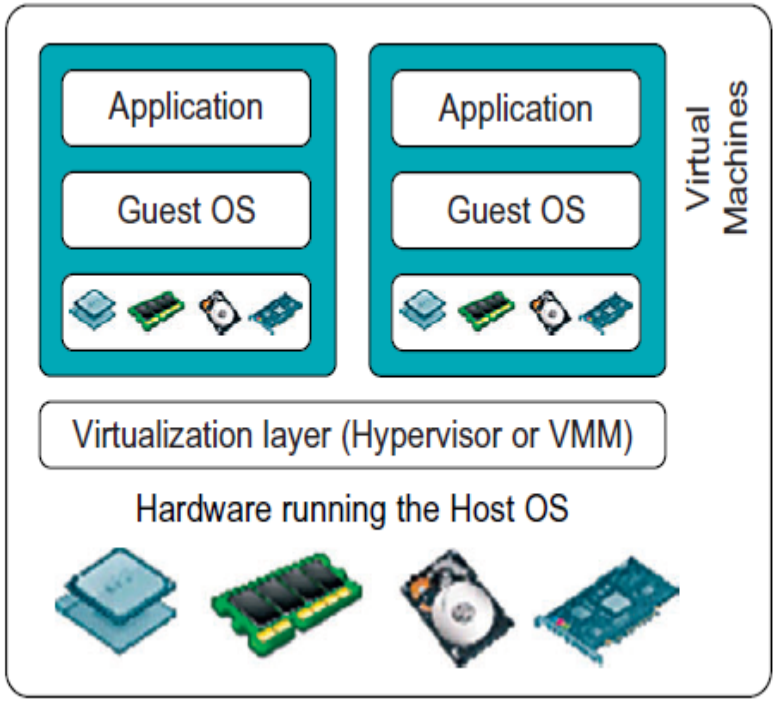
Preface

Content of this Lecture:

- In this lecture, we will discuss virtualization technology its importance, benefits, different models and key approaches to virtualization in CPU, Memory and Device Virtualization.

What is Virtualization ?

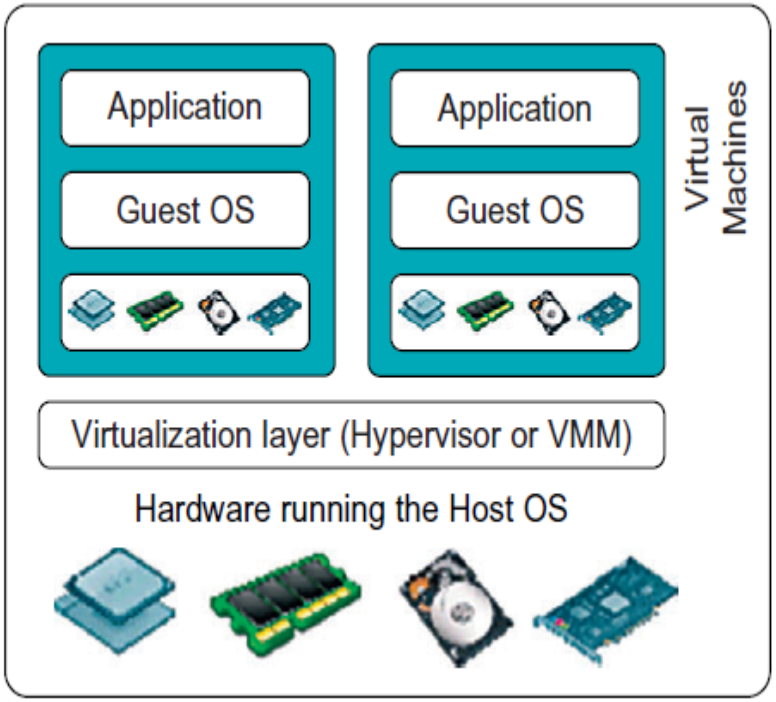
- Virtualization is originated in the 1960s at IBM.
- Virtualization **allows** concurrent execution of multiple OSs (and their applications) on the same physical machine.
- Virtual resources = **each OS thinks that it "owns" hardware resources**
- Virtual machine (VM) = **OS+ applications + virtual resources** (**guest domain**)
- Virtualization layer = **management of physical hardware** (**virtual machine monitor, hypervisor**)



Defining Virtualization

- A virtual machine is an **efficient, isolated duplicate of the real machine.**

- Supported by a **virtual machine monitor (VMM):**
  1. Provides environment **essentially identical with the original machine**
  2. Programs show **at worst only minor decrease in speed.**
  3. VMM is in **complete control of system resources.**
     (This means that the virtual machine monitor has full control to make decisions, who accesses which resources and when)



VMM Goals: Fidelity
Performance
Safety & isolation

Benefits of Virtualization

- **Consolidation:** It is this ability to run multiple virtual machines, with their operating systems and applications on a single physical platform.

  - Decrease cost, improve manageability (with fewer admins and with fewer electrical bills)

- **Migration:** Migrate the OS in the applications from one physical machine to another physical machine.

  - Greater availability of the services, improve reliability

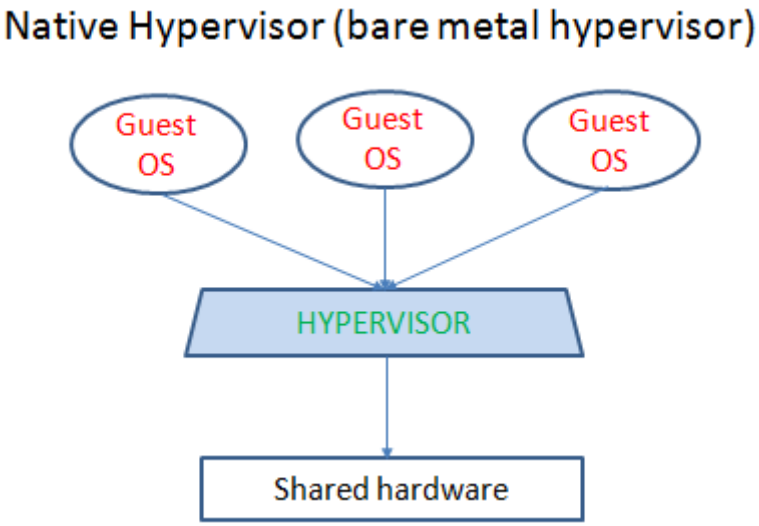
- **Security:** As the OS and the applications are nicely encapsulated in a virtual machine. It becomes more easy to contain any kinds of bugs, or any kinds of malicious behavior, to those resources that are available to the virtual machine only, and not to potentially affect the entire hardware system.

- **Some other benefits:** Debugging, Provide affordable Support for legacy OSs

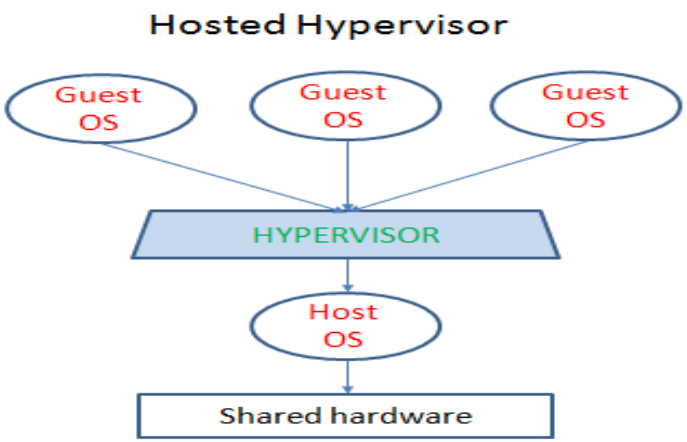
Virtualization Models

The two popular models for virtualization are called:

1. Bare-metal hypervisor
or Native Hypervisor (Type 1)



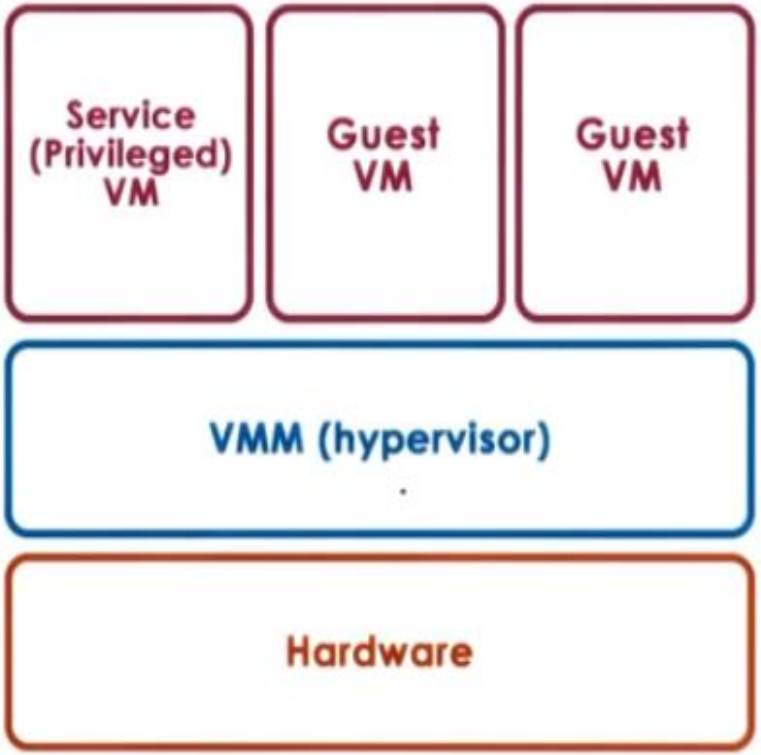
2. Hosted Hypervisor (Type 2)



Bare-metal virtualization model

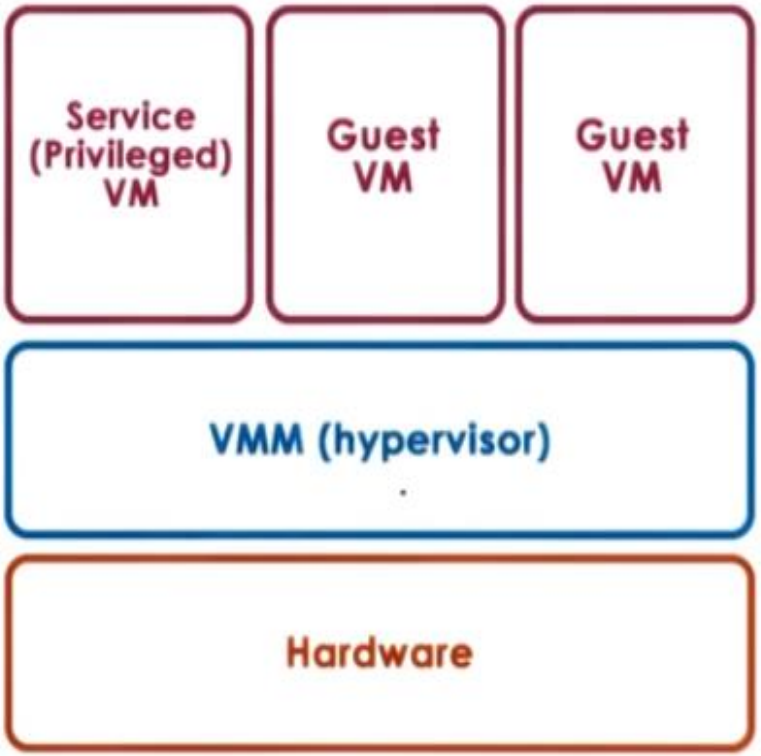
Bare-metal hypervisor (Type 1)

- **VMM (hypervisor)** manages all hardware resources and supports execution of entire VMs.
- **Privileged, service VM** to deal with devices (and other configuration and management task)



Bare-metal virtualization model

- This model is adapted by the **Xen** virtualization solution (open source or Citrix Xen Server) and also by the VMware's hypervisor, the **ESX hypervisor.**



(i) Xen (Open source or Citrix Xen Server)

- The VMs that are run in the virtualized environment are referred to as domains.
- The privileged domain is called dom 0, and the guest VMs are referred to as domUs.
- Xen is the actual hypervisor and all of the drivers are running in the privileged domain, in dom 0.

(ii) ESX (VMware)

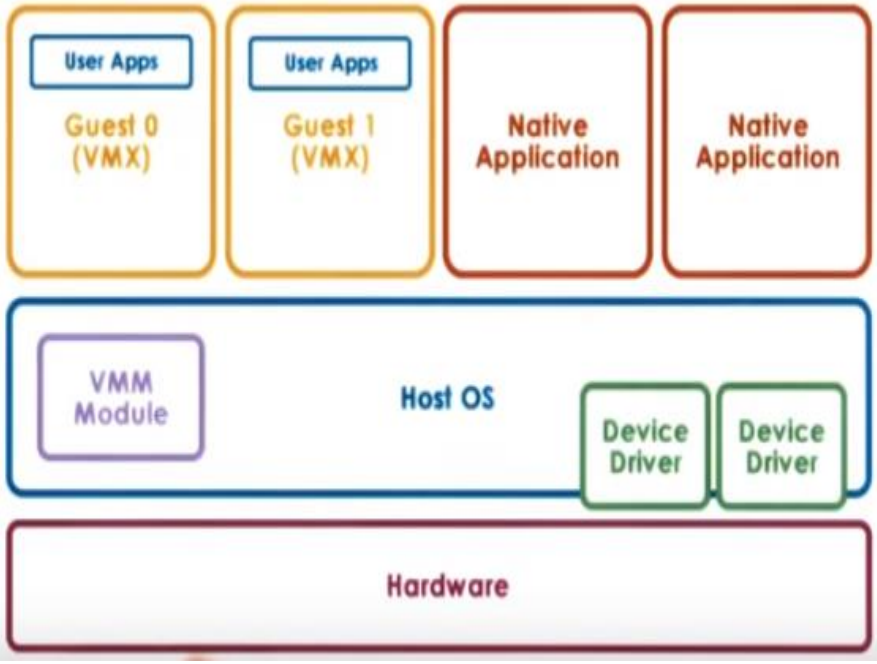
-Given that VMware and its hypervisors were first to market, VMware still owns the largest percentage of virtualized server cores. So these server cores run the ESX hypervisor and also provide the drivers for the different devices. That are going to be part of the hypervisor. To support a third party community of developers VMware exports a number of APIs.

Hosted virtualization model

Hosted Hypervisor (Type 2)

- In this model, at the lowest level, there is a **full fledged host OS that manages all of the hardware resources.**

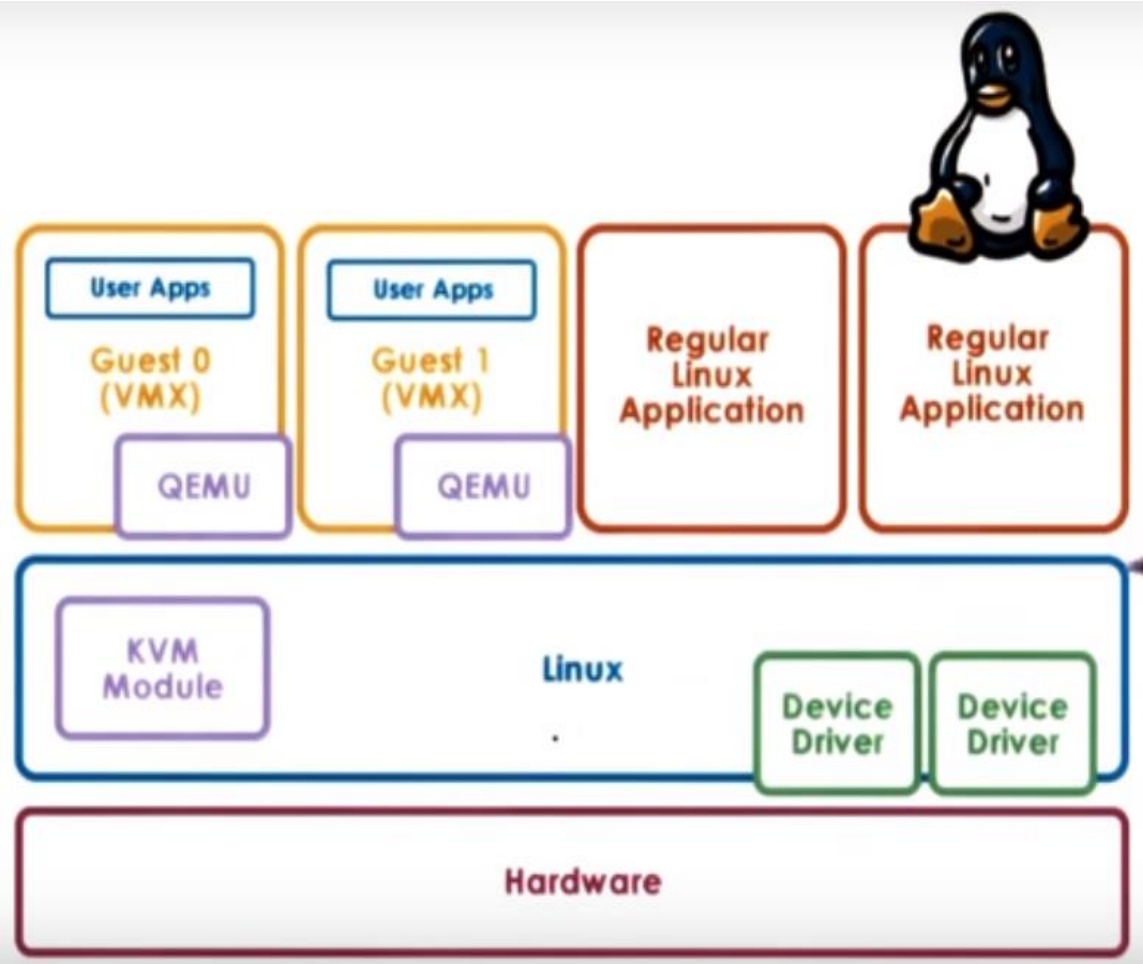
- The Host OS integrates a VMM module, that's responsible for providing the virtual machines with their virtual platform interface and for managing all of the context switching scheduling, etc.



Hosted virtualization model

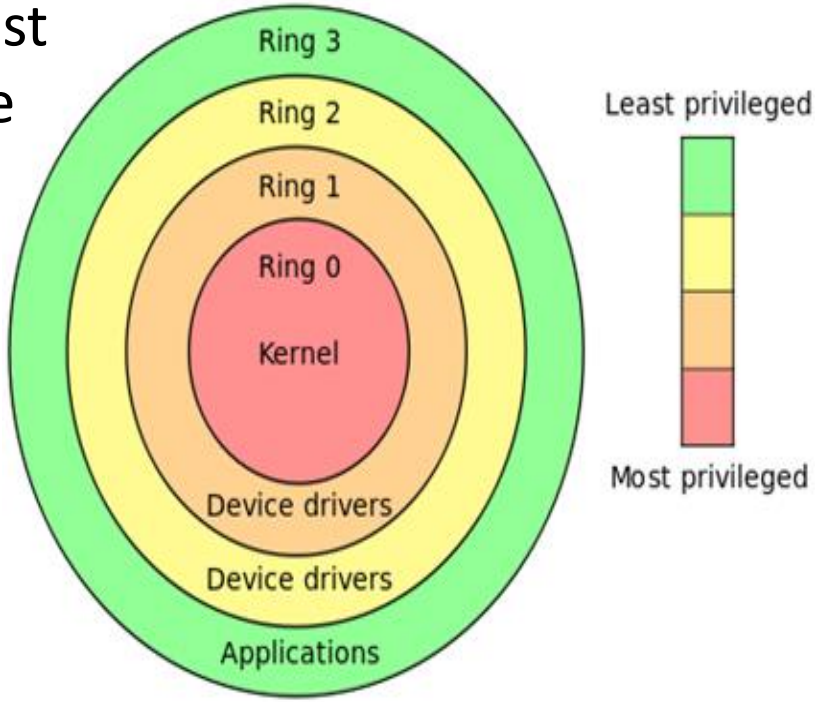
Example:

- **KVM** (Kernel-based VM)
- Based on Linux
- KVM kernel module + hardware emulator called QEMU for hardware virtualization
- Leverages large Linux open-source community



Hardware Protection Levels

- Commodity hardware actually has more than two protection levels.
- E.g. x86 architecture has four protection levels called **rings.**
- **Ring 3:** In contrast, ring 3 has the least level of privilege, so this is where the applications would reside.
- **Ring 0:** has the highest privilege and can access all of the resources and execute all hardware-supported instructions.



x86 Hardware without Virtualization

- The **x86** architecture offers **four levels** of privilege known as Ring 0, 1, 2 and 3 to operating systems and applications to manage access to the computer hardware. While user level applications typically run in Ring 3, the operating system needs to have direct access to the memory and hardware and must execute its privileged instructions in Ring 0.

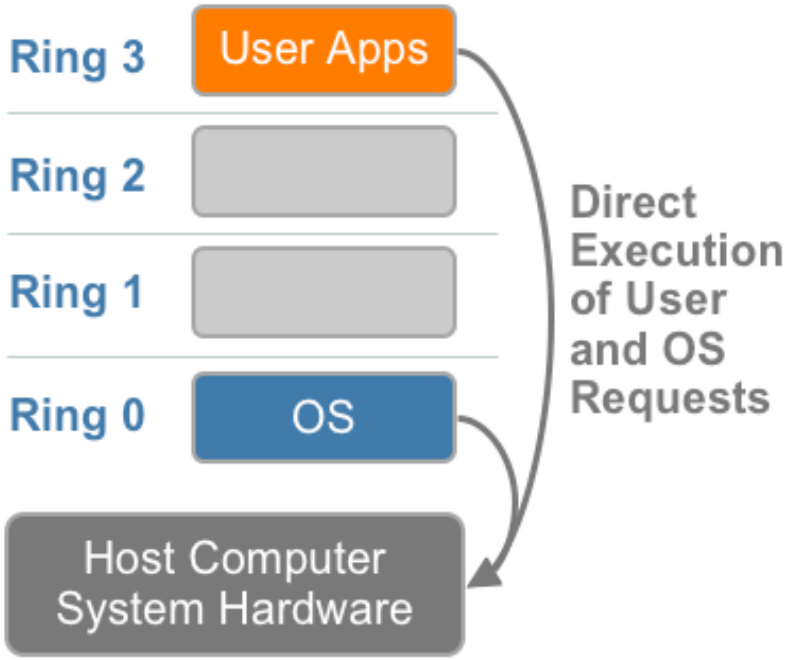


Fig - x86 privilege level architecture without virtualization

Processor Virtualization (Trap-and-Emulate)

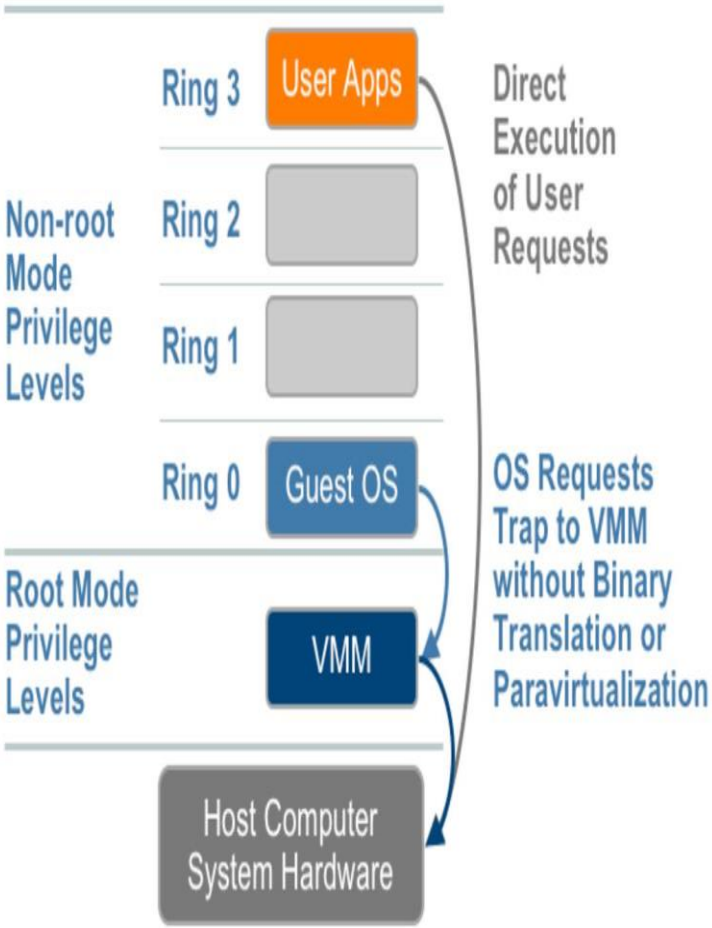
- Guest instructions
  - -executed **directly by hardware**
  - -for non-privileged operations: hardware speeds=> efficiency
  - -for privileged operations: trap to hypervisor

- Hypervisor determines what needs to be done:
- **If illegal operation:** terminate VM
- **If legal operation:** emulate the behavior the guest OS was expecting from the hardware

Problems with Trap-and-Emulate

- X85 Pre 2005
- -4 rings, no root/non-root modes yet
- -hypervisor in ring 0, guest OS in ring 1
- BUT: 17 privileged instructions do not trap! Fail silently!

  E.g. interrupt enable/disable bit in privileged register; POPF/PUSHF instructions that access it from ring 1 fail silently
- Hypervisor doesn't know, so it doesn't try to change settings
- OS doesn't know, so assumes change was successful/



Binary Translation

- **Main idea:** rewrite the VM binary to never issue those 17 instructions
- Pioneered by Mendel Rosenblum's group at Stanford, commercialized as Vmware
- Rosenblum awarded ACM Fellow for "reinventing virtualization"

Binary Translation

Binary translation:

- **Goal:** full virtualization=guest OS not modified
- **Approach:** dynamic binary translation

1. Inspect code blocks to be executed
2. If needed, translate to alternate instruction sequence

-e.g., to emulate desired behavior, possibly even avoiding trap

3. Otherwise, run at hardware speeds

-Cache translated blocks to amortize translation costs

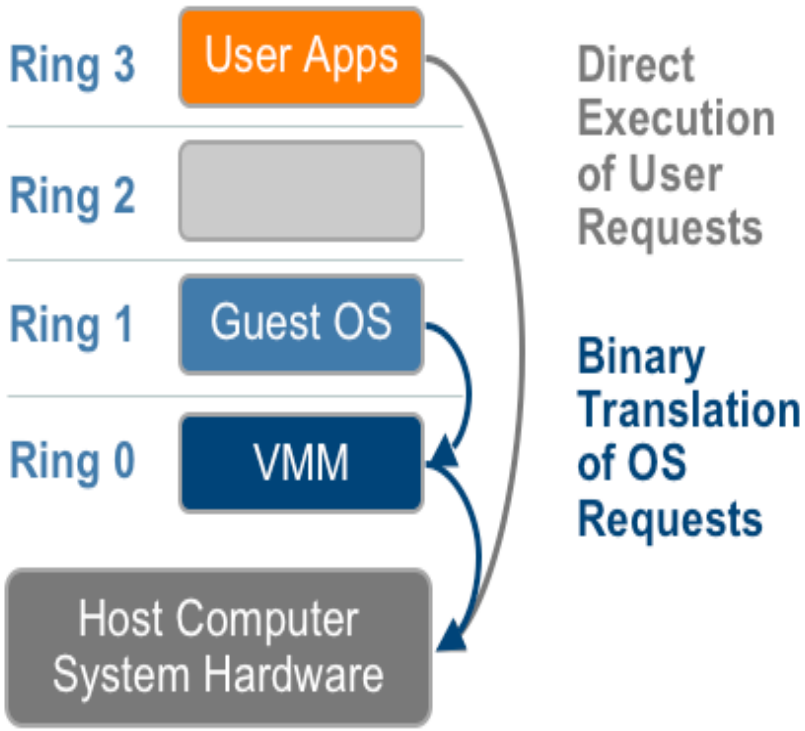
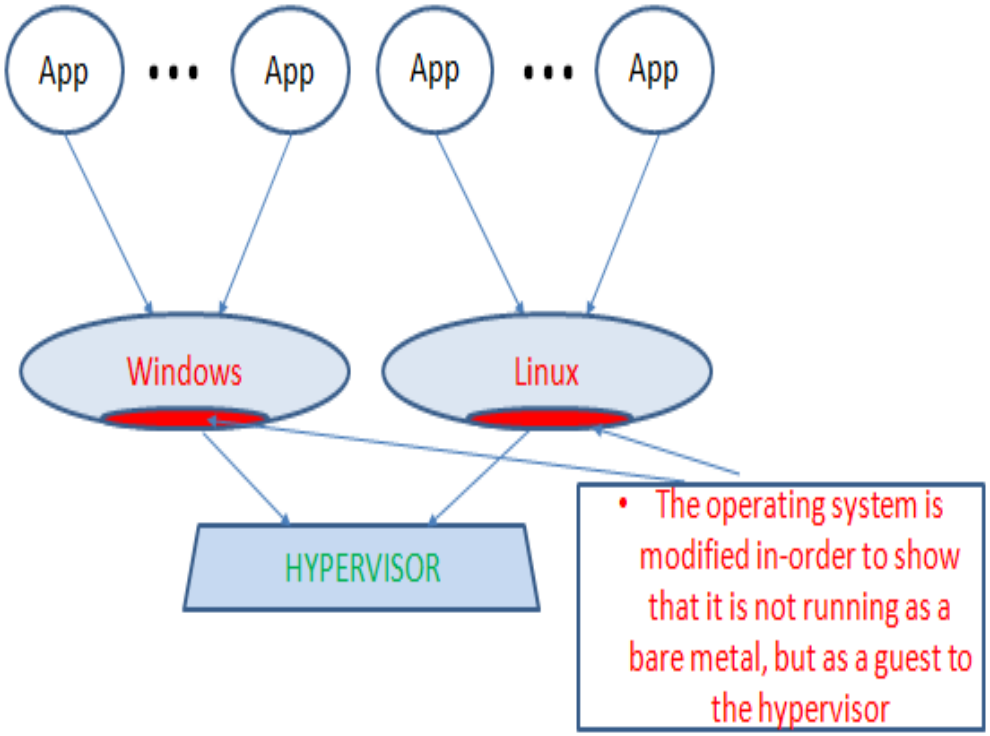


Fig.: Binary translation approach to x86 virtualization

Para virtualization

- **Goal:** performance; give up on unmodified guests
- Approach: Para virtualization =modify guest so that
- It knows it's running virtualized
- It makes explicit calls to the hypervisor (hypercalls)
- Hypercall (~system calls)
  - Package context info
  - Specify desired hypercall
  - Trap to VMM
  - e.g. Xen= open source hypervisor (XenSource -> Citrix)



Para virtualization: Review !

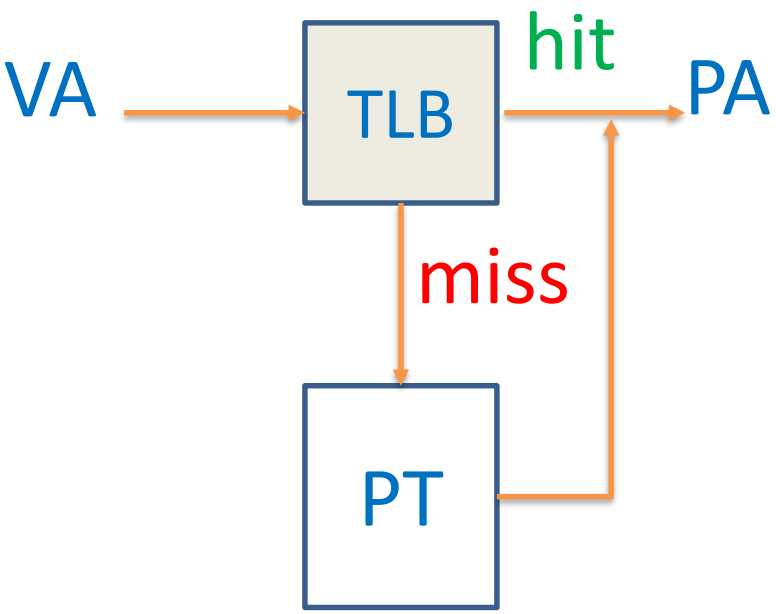
- What percentage of Guest OS code may need modification with para Virtualization?
  - 10 %
  - 50%
  - 30%
  - 2%
- Answer: less than 2%.
  - This can be shown by a proof-of-construction by XEN
  - Xen is a para virtualized hypervisor.

Memory Virtualization

- To run multiple virtual machines on a single system, one has to virtualize the **MMU**(memory management unit) to support the guest OS.
- The **VMM** is responsible for mapping guest physical memory to the actual machine memory, and it uses **shadow page tables** to accelerate the mappings.
- The VMM uses **TLB** (translation lookaside buffer) hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access.

Shadow Page table

- VA- Virtual Address
- PA- Physical Address



- CPU uses PT for address translation
- Hardware PT is really the S PT(shadow Page Table)

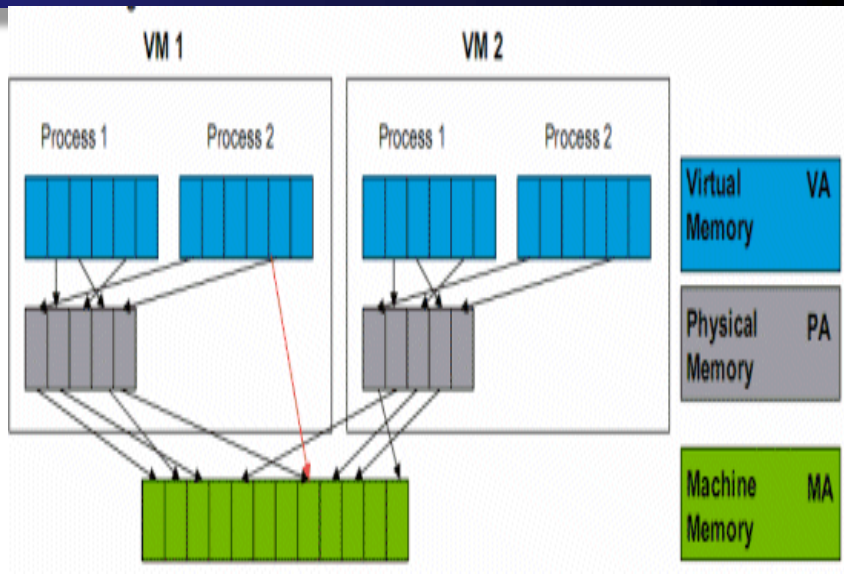
Memory Virtualization

- **Full virtualization**

  -all guests expect contiguous physical memory, starting at 0

  -virtual vs. physical vs. machine addresses and page frame numbers

  -Still leverages hardware MMU, TLB



Option 1:

-Guest page table: VM => PA

-Hypervisor: PA=> MA

-Too expensive

Option 2:

-Guest page table: VA => PA

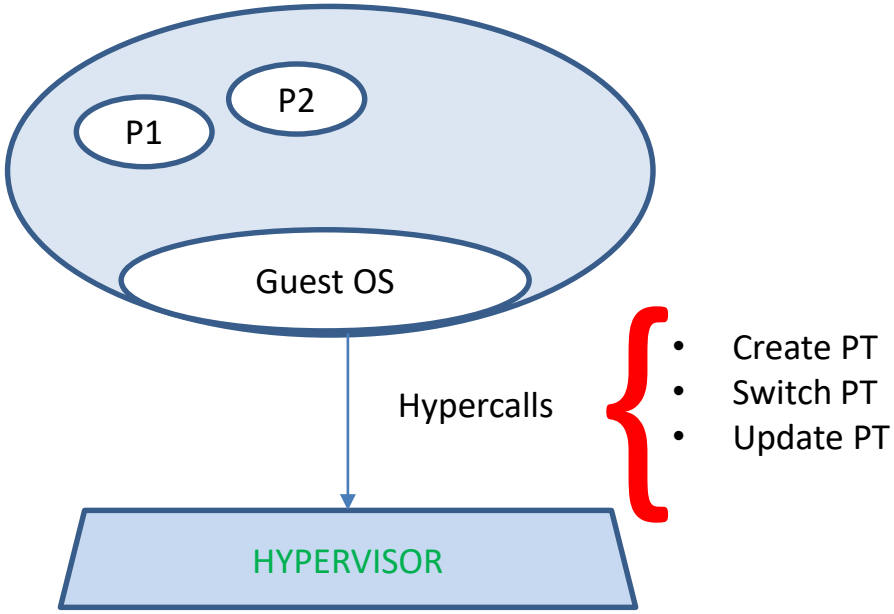
-Hypervisor shadow Page Table: VA=> MA

-Hypervisor maintains consistence

e.g. invalidate on ctx switch, write-protect guest PT to track new mappings

Memory Virtualization

- **Para virtualization**
  - -Guest aware of virtualization
  - -No longer strict requirement on contiguous physical memory starting at 0
  - -Explicitly registers page tables with hypervisor
  - -Can "batch" page table updates to reduce VM exists
  - -Other optimizations



- Overheads eliminated or reduced on newer platforms

Device Virtualization

- For CPUs and memory
- Less diversity, ISA (instruction set architectures)level standardization of interface
- For devices
- High diversity
- Lack of standard specification of device interface and behavior
- Three Key models for device virtualization

  (i) Passthrough Model

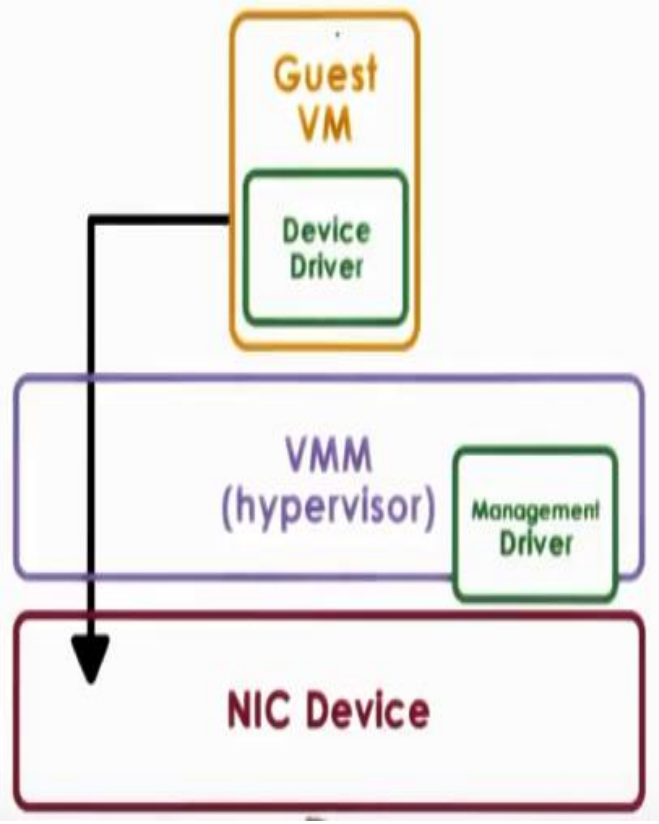
  (ii) Hypervisor Direct Model

  (iii) Split Device Driver Model

(i) Passthrough Model

Approach: VMM-level driver configures device access permissions

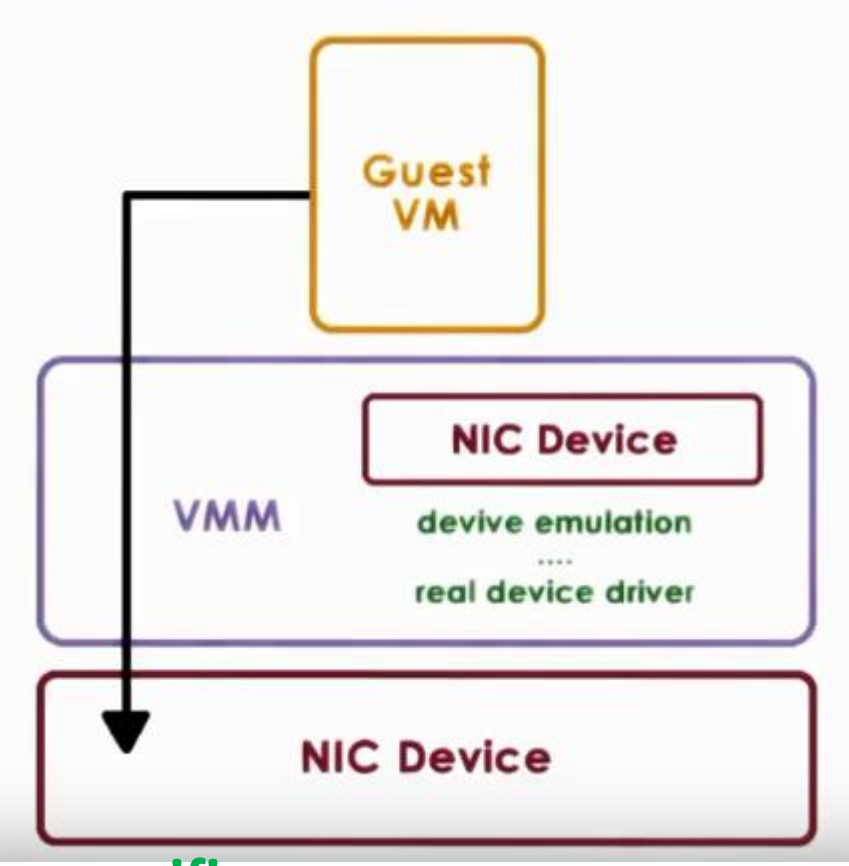
- VM provided with exclusive access to the device
- VM can directly access the device (also called VMM-bypass model)
- Device sharing will become difficult.
- VMM must have exact type of device as what VM expects
- VM migration is tricky



(ii) Hypervisor-Direct Model

Approach:

- VMM intercepts all device accesses
- Emulate device operation:
  - translate to generic I/O operation
  - traverse VMM-resident I/O stack
  - invoke VMM-resident driver



Key benefits:

VM decoupled from physical device

Sharing, migration, dealing with device specifies

Downside of the model

Latency of device operations

Device driver ecosystem complexities in hypervisor

(iii) Split-Device Driver Model

Approach:

Device access control **split** between

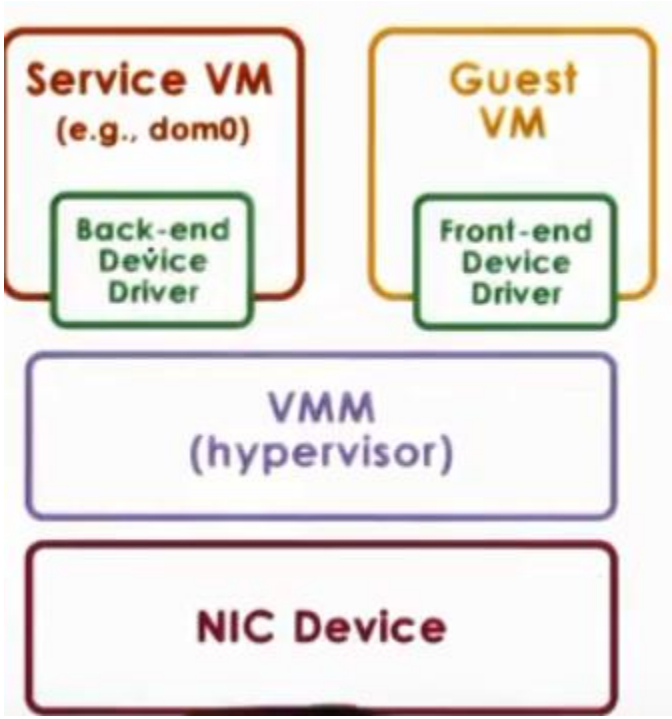
-Front end driver in guest VM (device API)

-Back-end driver in service VM (or host)

Modified guest drivers

i.e. Limited to para virtualized guests

Eliminate emulation overhead allow for better management of shared devices



Conclusion

- In this lecture, we have **defined virtualization** and discussed the main virtualization approaches.

- We have also described **processor virtualization, memory virtualization and device virtualization** used in virtualization solutions, such as Xen, KVM and the VMware .