

---

# Cloud Computing

## CS9243 Guest Lecture

Hiroshi Wada, PhD

Jorke Odolphi

Software Systems Research Group, NICTA  
Yuruware

# Agenda

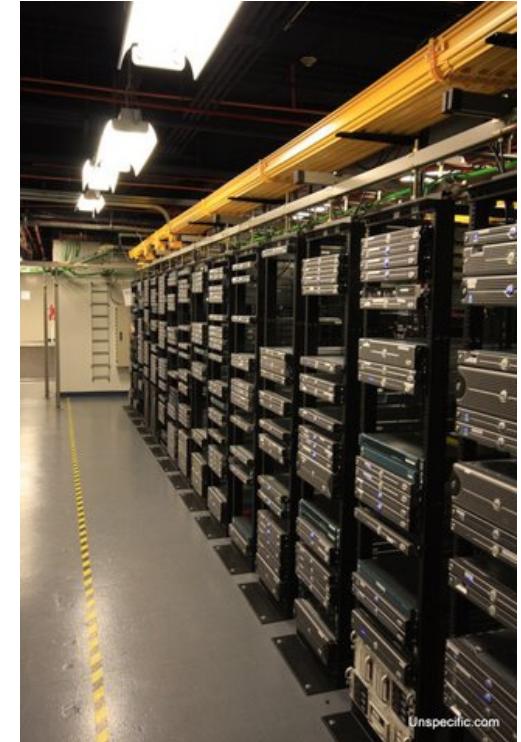
---

- Introduction to cloud
  - Why Cloud Computing?
  - Look into Amazon Web Service and Google App Engine
- “Real” Cloud Architecture
- Cloud Architecture and NoSQL
- Application Architecture on Cloud
  - Scaling and fault tolerance

# Operating infrastructure is not easy



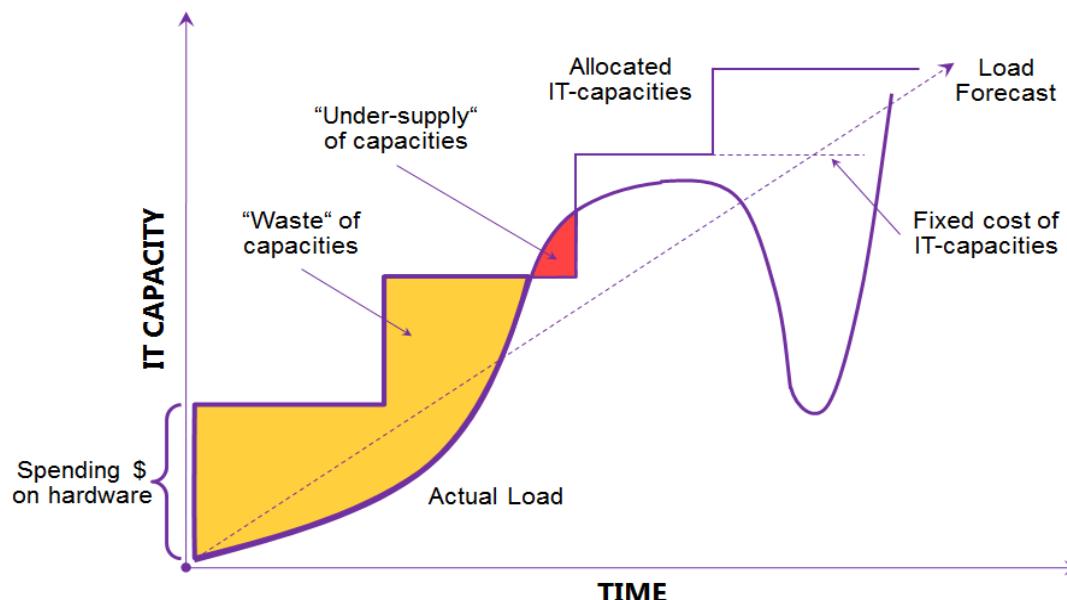
- Large upfront and running cost
  - Facility, hardware and configuration
  - 24h monitoring, backup, hardware maintenance
- Capacity planning is not easy
  - You cannot change infrastructure quickly
    - Not easy to change the facility design
    - Often takes days or weeks to obtain new hardware
  - Predict the peak load and provision based on it
    - e.g., 14:00 on Friday, Christmas, tax returns, ...



<http://www.flickr.com/photos/neospire/3594831687/>

# Over and under provisioning

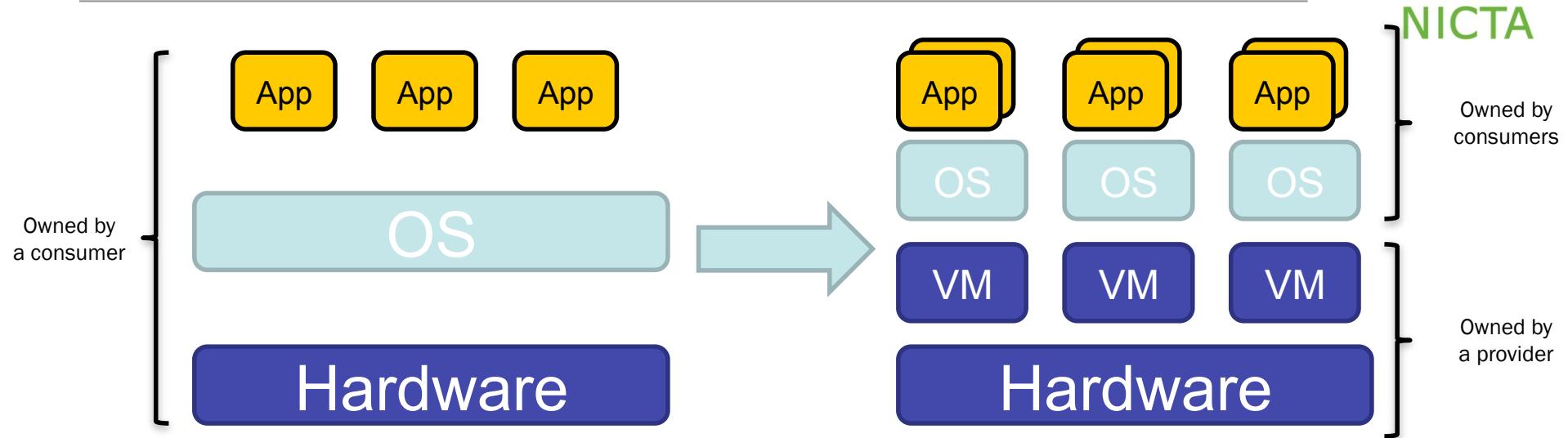
- “overly” over provision is not uncommon due to the fear of under provisioning and lack of flexibility
- Issues
  - Waste of resources
    - In large data centers, average usage of servers is 5% – 15%
  - Cannot decommission easily to follow the workload
  - Cannot handle sudden workload spikes



# Cloud Computing



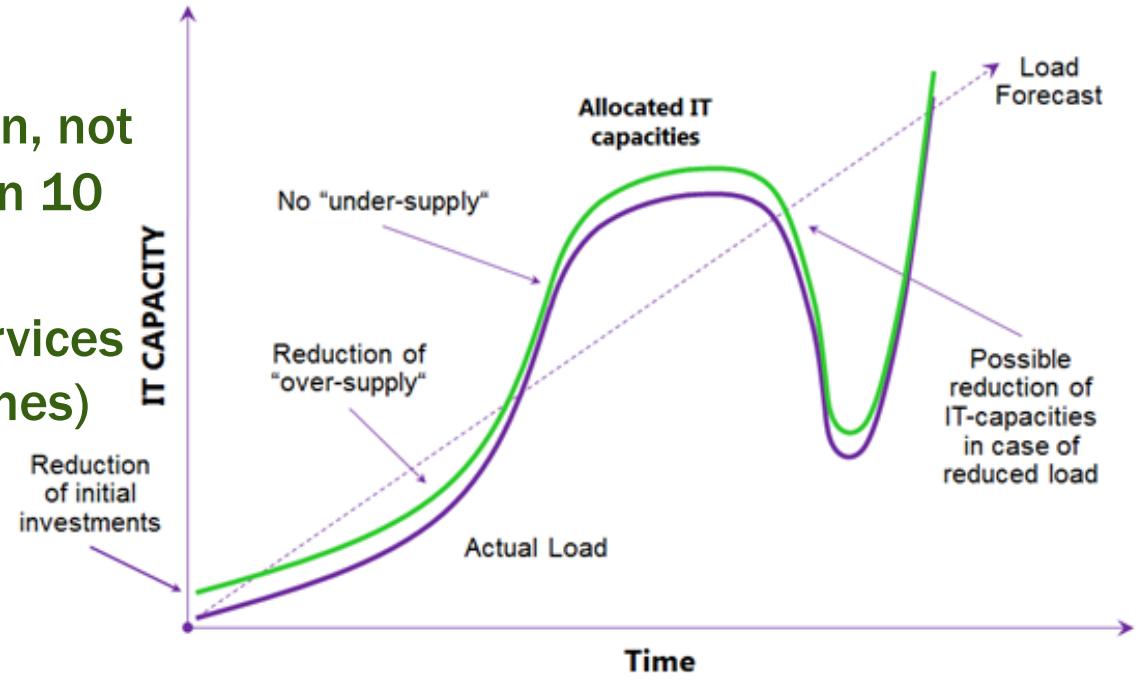
NICTA



- Basic idea is to share computing resources among multiple users with proper isolation
  - e.g., Split a physical server into multiple virtual machines
- Providers can reduce the cost per resource unit
  - Bulk purchase lowers IT equipment cost
  - Managing 1000 servers costs same as managing 20 servers
- Consumers can buy resources only when needed

# Why Cloud Computing

- **High Elasticity/Scalability**
  - Virtually infinite amount of resources is available on demand
  - Cost of 1 machine for 100 hours = Cost of 100 machines for 1 hour
- **(Possibly) Reduce cost and complexity**
  - Pay per use and no up-front cost
  - No in-house data center management
- **Ease-of-use**
  - Obtain a server in 5 min, not days, and throw away in 10 sec
  - Various on-demand services (not only virtual machines)



# Leading Provider: Amazon EC2

---



**Let's see how Amazon EC2, a leading commercial cloud, looks**



**1. Grab your credit card  
and create an account.  
(10 min) Then, access  
to a console**



The screenshot shows the Amazon EC2 Dashboard. On the left, a sidebar lists services: EC2 Dashboard, Events, Instances, AMIs, Elastic Block Store, and Network & Security. The main area displays "Resources" with a summary of current usage: 1 Running Instance, 1 Volume, 2 Key Pairs, 0 Elastic IPs, 1 Snapshot, 0 Load Balancers, and 2 Security Groups. Below this is a "Create Instance" section with a "Launch Instance" button. A note states: "Note: Your instances will launch in the Asia Pacific (Sydney) region". To the right, there's a "Scheduled Events" section showing "No events" for the "Asia Pacific (Sydney)" region. At the top right, a dropdown menu shows "Sydney" selected from a list of regions: US East (N. Virginia), US West (Oregon), US West (N. California), EU (Ireland), Asia Pacific (Singapore), Asia Pacific (Tokyo), and Asia Pacific (Sydney). A red arrow points from the "3. Hit this button" callout to the "Launch Instance" button. Another red arrow points from the "2. Select where you want to create your virtual machines (called 'instance')" callout to the "Asia Pacific (Sydney)" region in the dropdown menu.

**3. Hit this button**

**2. Select where you want to create your virtual machines (called 'instance')**

east-2

Hiroshi Wada | Sydney | Help

EC2 Dashboard

Events

INSTANCES

Instances

Spot Requests

Reserved Instances

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

NETWORK & SECURITY

Security Groups

Elastic IPs

Placement Groups

Load Balancers

Key Pairs

Network Interfaces

Resources

You are using the following Amazon EC2 resources in the Asia Pacific (Sydney) region:

1 Running Instance      0 Elastic IPs  
1 Volume      1 Snapshot  
2 Key Pairs      0 Load Balancers  
Placement Groups not supported in this region      2 Security Groups  
Supported Platforms: EC2-Classic, EC2-VPC

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

**Launch Instance**

Note: Your instances will launch in the Asia Pacific (Sydney) region

Service Health

Service Status:  Asia Pacific (Sydney)

Availability Zone S

ap-southeast-2a Availability zone is operating normally  
 ap-southeast-2b Availability zone is operating normally

Scheduled Events

Asia Pacific (Sydney)  
No events

Feedback

Ad

US East (N. Virginia)

Gett

Doc

All E

Find

Foru

Pric

Fee

Fee

Report an issue

US West (Oregon)

US West (N. California)

EU (Ireland)

Asia Pacific (Singapore)

Asia Pacific (Tokyo)

Asia Pacific (Sydney)

South America São Paulo

## 4. Select a machine image (called AMI) to use

- Various OS (Major distros, Redhat, Windows, ...)
  - Some are provided by official providers, some are by 3<sup>rd</sup> parties
- Many pre-configured images
  - e.g., CentOS with Apache + MySQL + lots, Windows + SQL Server, ...
- You can “save” your instance to create your own AMI

The screenshot shows the AWS EC2 Management Console interface. On the left, there's a sidebar with navigation links for EC2 Dashboard, Services, Instances, Images, Elastic Block Store, Network & Security, and more. The main area is titled 'Create a New Instance'.

The 'Create a New Instance' wizard has three options:

- Classic Wizard**: Launch an On-Demand or Spot instance using the classic wizard with fine-grained control over how it is launched.
- Quick Launch Wizard**: Launch an On-Demand instance using an editable, default configuration so that you can get started in the cloud as quickly as possible.
- AWS Marketplace**: AWS Marketplace is an online store where you can find and buy software that runs on AWS. Launch with 1-Click and pay by the hour.

On the right, a modal window titled 'Request Instances Wizard' is open, showing a list of available Amazon Machine Images (AMIs). The 'My AMIs' tab is selected, displaying several options:

- Basic 32-bit Amazon Linux AMI 2011.02.1 Beta (AMI Id: ami-8c1fecc5)
- Basic 64-bit Amazon Linux AMI 2011.02.1 Beta (AMI Id: ami-0e1fecc7)
- Red Hat Enterprise Linux 6.1 32 bit (AMI Id: ami-0cbh4265)
- Red Hat Enterprise Linux 6.1 64 bit (AMI Id: ami-5e837b37)
- SUSE Linux Enterprise Server 11 64-bit (AMI Id: ami-e4a3578d)

Each AMI entry includes a 'Select' button with a star icon. At the bottom of the modal, there's a note about free tier eligibility and a 'Continue' button.

At the bottom of the main page, there are links for 'Submit Feedback' and 'Getting Started Guide'. The footer contains copyright information: © 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use. There's also a 'Feedback' button.

## 5. Determine the amount of resources to allocate

- <1.0Ghz CPU + 600MB RAM → 0.02 USD/hour
- ~1.0Ghz CPU + 1.7GB RAM → 0.08 USD/hour
- ~3.0Ghz x 8 cores + 68GB RAM → 2.0 USD/hour
- You can pay Win/SQL Server license fees in pay-per-hour

The screenshot shows the 'Request Instances Wizard' dialog box over a blurred EC2 Dashboard background. The dashboard sidebar includes links for EC2 Dashboard, Events, Instances, AMIs, Elastic Block Store, Network & Security, and more.

The 'INSTANCE DETAILS' tab is selected in the wizard. It displays the following information:

- Number of Instances: 1
- Instance Type: T1 Micro (t1.micro, 613 MiB)
- Launch as an EBS-backed instance (radio button selected):
  - T1 Micro (t1.micro) ★ Free tier eligible
  - M1 Small (m1.small)
  - M1 Medium (m1.medium)
  - M1 Large (m1.large)
  - M1 Extra Large (m1.xlarge)
  - M2 High-Memory Extra Large (m2.xlarge)
  - M2 High-Memory Double Extra Large (m2.2xlarge)
  - M2 High-Memory Quadruple Extra Large (m2.4xlarge)
  - M3 Extra Large (m3.xlarge)
  - M3 Double Extra Large (m3.2xlarge)
  - C1 High-CPU Medium (c1.medium)
  - C1 High-CPU Extra Large (c1.xlarge)
- Launch into: [dropdown menu]
- Request Spot Instances (radio button): [disabled]

At the bottom of the wizard are 'Back' and 'Continue' buttons.

## 6. Done! (< 5 minutes in total)

- Set SSH key, configure firewall, etc.
- Each machine has a randomly assigned public IP address and DNS name. e.g.,  
**ec2-54-252-24-185.ap-southeast-2.compute.amazonaws.com**

The screenshot shows the AWS EC2 Management Console interface. On the left, there's a navigation sidebar with links for EC2 Dashboard, Events, Instances (selected), Spot Requests, Reserved Instances, Images (AMIs), Elastic Block Store (Volumes, Snapshots), Network & Security (Security Groups, Elastic IPs, Placement Groups, Load Balancers, Key Pairs, Network Interfaces). The main area shows a table of instances:

	Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitoring	Security Groups	Key
<input type="checkbox"/>	Cloudmon	i-	ami-e2ba2cd8	ebs	t1.micro	running	2/2 checks p none	none	basic	default	clo
<input checked="" type="checkbox"/>	empty	i-dfb75ae2	ami-e2ba2cd8	ebs	m1.small	running	initializing... none	none	basic	default	syc

A red arrow points from the text "I got my virtual machine!" to the selected instance row. The modal window below shows the details for the selected instance:

**1 EC2 Instance selected.**

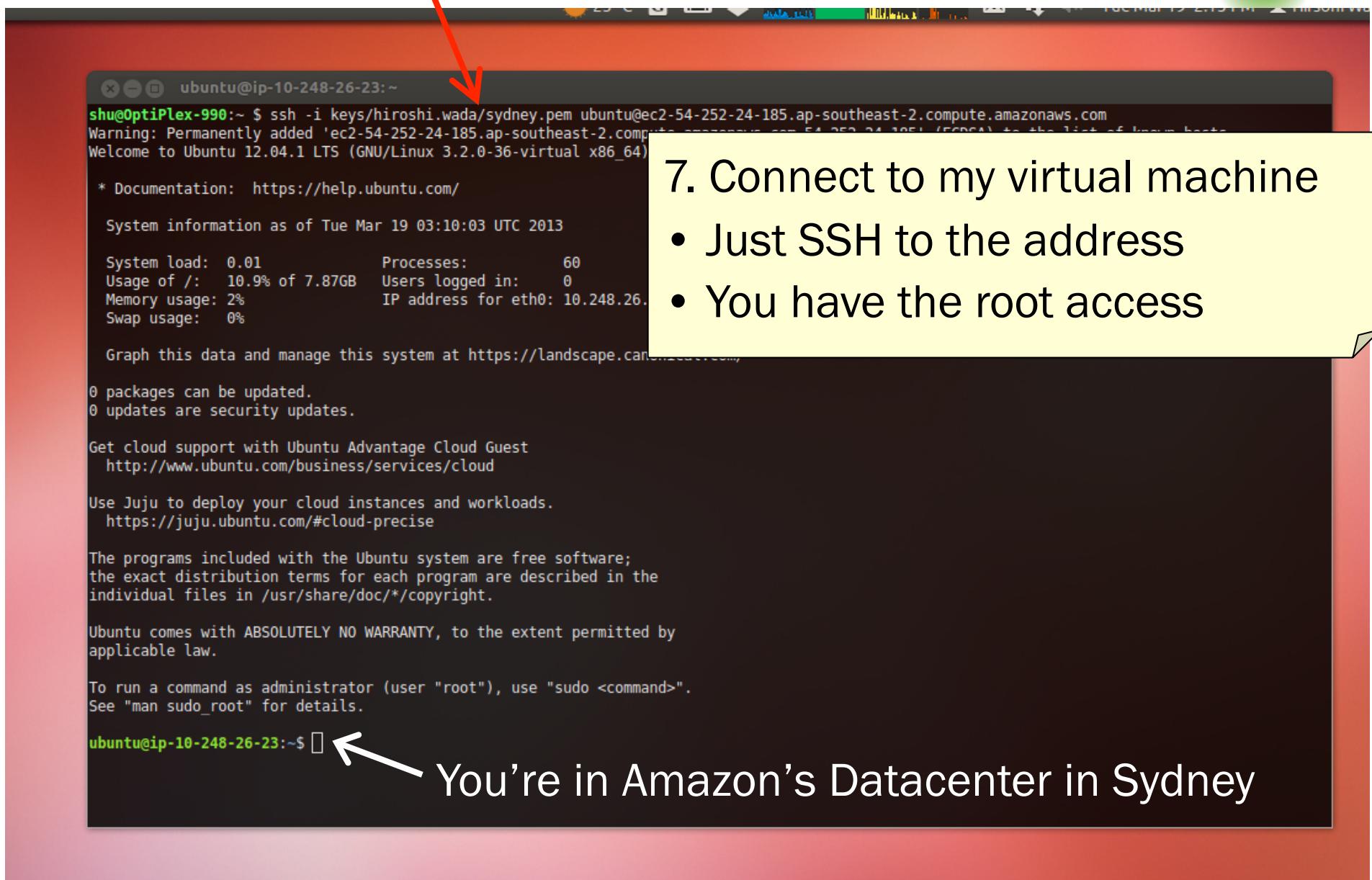
**EC2 Instance: i-dfb75ae2** ●  
[ec2-54-252-24-185.ap-southeast-2.compute.amazonaws.com](https://ec2-54-252-24-185.ap-southeast-2.compute.amazonaws.com)

Description	Status Checks	Monitoring	Tags
AMI: ubuntu/images/ebs/ubuntu-precise-12.04-amd64-server-20130124 (ami-e2ba2cd8)			
Zone: ap-southeast-2b			
Type: m1.small			
Scheduled Events: No scheduled events			
VPC ID: -			
Source/Dest. Check: -			
Placement Group: -			
RAM Disk ID: -			
Key Pair Name: sydney			

Alarm Status:	none
Security Groups:	default. view rules
State:	running
Owner:	309657489351
Subnet ID:	-
Virtualization:	paravirtual
Reservation:	r-98ebf3a2
Platform:	-
Kernel ID:	aki-31990e0b

At the bottom, there are links for © 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use, and a Feedback button.

# SSH from a desktop



A screenshot of a Linux desktop environment showing a terminal window. The terminal window has a red arrow pointing to its title bar which reads "ubuntu@ip-10-248-26-23:~". Another white arrow points to the command line prompt "ubuntu@ip-10-248-26-23:~\$". The terminal content shows an SSH session to an AWS instance:

```
shu@OptiPlex-990:~ $ ssh -i keys/hiroshi.wada/sydney.pem ubuntu@ec2-54-252-24-185.ap-southeast-2.compute.amazonaws.com
Warning: Permanently added 'ec2-54-252-24-185.ap-southeast-2.compute.amazonaws.com' (RSA) to the list of known hosts.
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-36-virtual x86_64)

 * Documentation:  https://help.ubuntu.com/
 
 System information as of Tue Mar 19 03:10:03 UTC 2013

 System load:  0.01      Processes:          60
 Usage of /:   10.9% of 7.87GB  Users logged in:    0
 Memory usage: 2%           IP address for eth0: 10.248.26.
 Swap usage:   0%
 
 Graph this data and manage this system at https://landscape.canonical.com/...
 
 0 packages can be updated.
 0 updates are security updates.

Get cloud support with Ubuntu Advantage Cloud Guest
 http://www.ubuntu.com/business/services/cloud

Use Juju to deploy your cloud instances and workloads.
 https://juju.ubuntu.com/#cloud-precise

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-248-26-23:~$
```

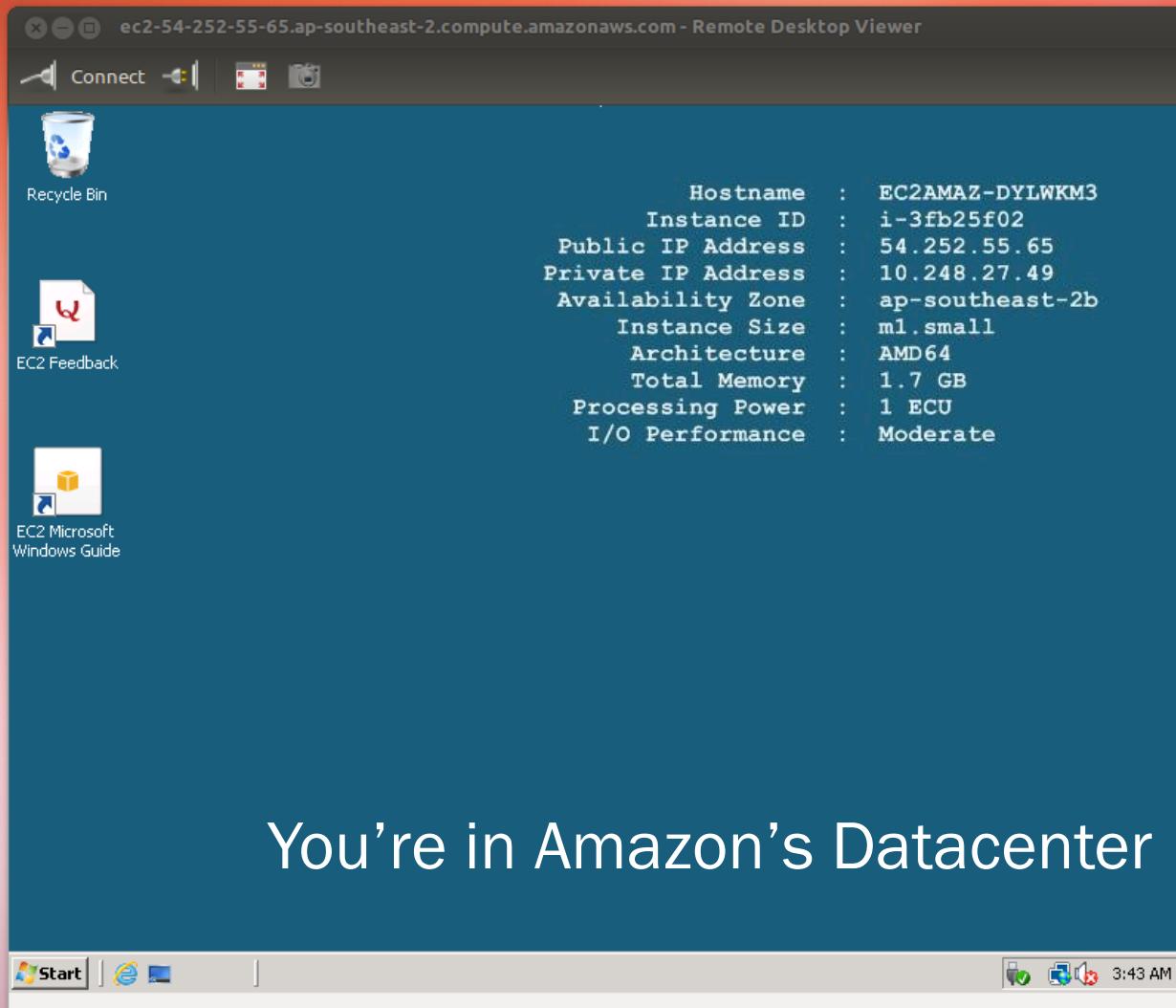
The terminal window is highlighted with a yellow box containing the following text:

7. Connect to my virtual machine

- Just SSH to the address
- You have the root access

You're in Amazon's Datacenter in Sydney

If you need Windows,  
launch a Windows instance  
and remote-desktop to it



**EC2 Management Console**

<https://console.aws.amazon.com/ec2/home?region=ap-southeast-2#s=Instances>

Hiroshi Wada | Sydney | Help

**Launch Instance** Actions

Viewing: All Instances All Instance Types Search

1 to 3 of 3 Instances

Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitoring	Security Groups	Key
Cloudmon	i-	ami-e2ba2cd8	ebs	t1.micro	running	2/2 checks p	none	basic	default	clo
empty	i-dfb75ae2	ami-e2ba2cd8	ebs	m1.small	terminated		none	basic	default	syd
empty	i-3fb25f02	ami-c4ce5ffe	ebs	m1.small	terminated		none	basic	quick-start-1	syd

**8. Terminate (decommission) or stop (shutdown/hibernate) instances when they are not in use**

- Instances cost you by time – not by actual resource usage
- In a past project, we use a script to stop instances at 8:00PM
- Restart instances manually in the morning if necessary

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#) [Feedback](#)

- Hours to run virtual machines
- Network in/out
- VPN
- Disk access
- # of requests made to services

# Programmable Infrastructure



```
https://ec2.amazonaws.com/?Action=RunInstances  
&ImageId=ami-xxxxxxx&InstanceType=m1.large  
&MaxCount=1&MinCount=1  
&KeyName=web&AUTHPARAMS
```

- All actions have REST APIs
- What you can do on the web interface can be done through REST as well
- Use of APIs opens the possibility of flexible operations
  - Obtain resources on demand
  - Release unused resources
  - Automate the environment building
  - Micro-rebooting of apps for error handling
  - ... will discuss later

```
$ ec2-run-instances ami-xxxxxx -g default -k web  
$ ec2-run-terminate i-xxxxxx
```

```
try {  
    RunInstancesRequest req = new RunInstancesRequest();  
    req.setImageId("ami-xxxx");  
    req.setPlacement(new Placement().withAvailabilityZone(az));  
    req.setSecurityGroupIds(securityGroupIds);  
    RunInstancesResult res = ec2client.runInstances(req);  
} catch (...)
```

- **SDKs are available for major programming languages**
  - iOS, Java, PHP, Ruby, ...
  - It's possible to write code with command line tools and sh
  - You can use curl to send REST messages ☺

# Five Characteristics – NIST Definition

---



- **On-demand Self-Service**
  - A consumer can provision computing capabilities without human interaction
- **Broad network access**
  - Computing capabilities are available over the network and accessed through standard mechanisms
- **Resource pooling**
  - Provider's computing resources are pooled to serve multiple consumers with different resources dynamically assigned according to consumers' demands
- **Rapid elasticity**
  - Computing capabilities can be rapidly and elastically provisioned to quickly scale out and rapidly released to scale in
- **Measured service**
  - Resource usage can be monitored, controlled, and reported. Providing transparency for both the provider and consumer

# Three Service Models – NIST definition



Technology exposed to customers



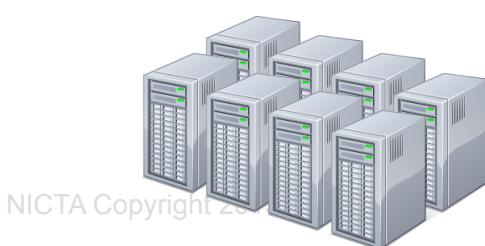
**Software  
as a Service**



**Platform  
as a Service**



**Infrastructure  
as a Service**



Datacenter  
Infrastructure  
From imagination to impact

Commercial providers



**salesforce.com®**  
Success On Demand.



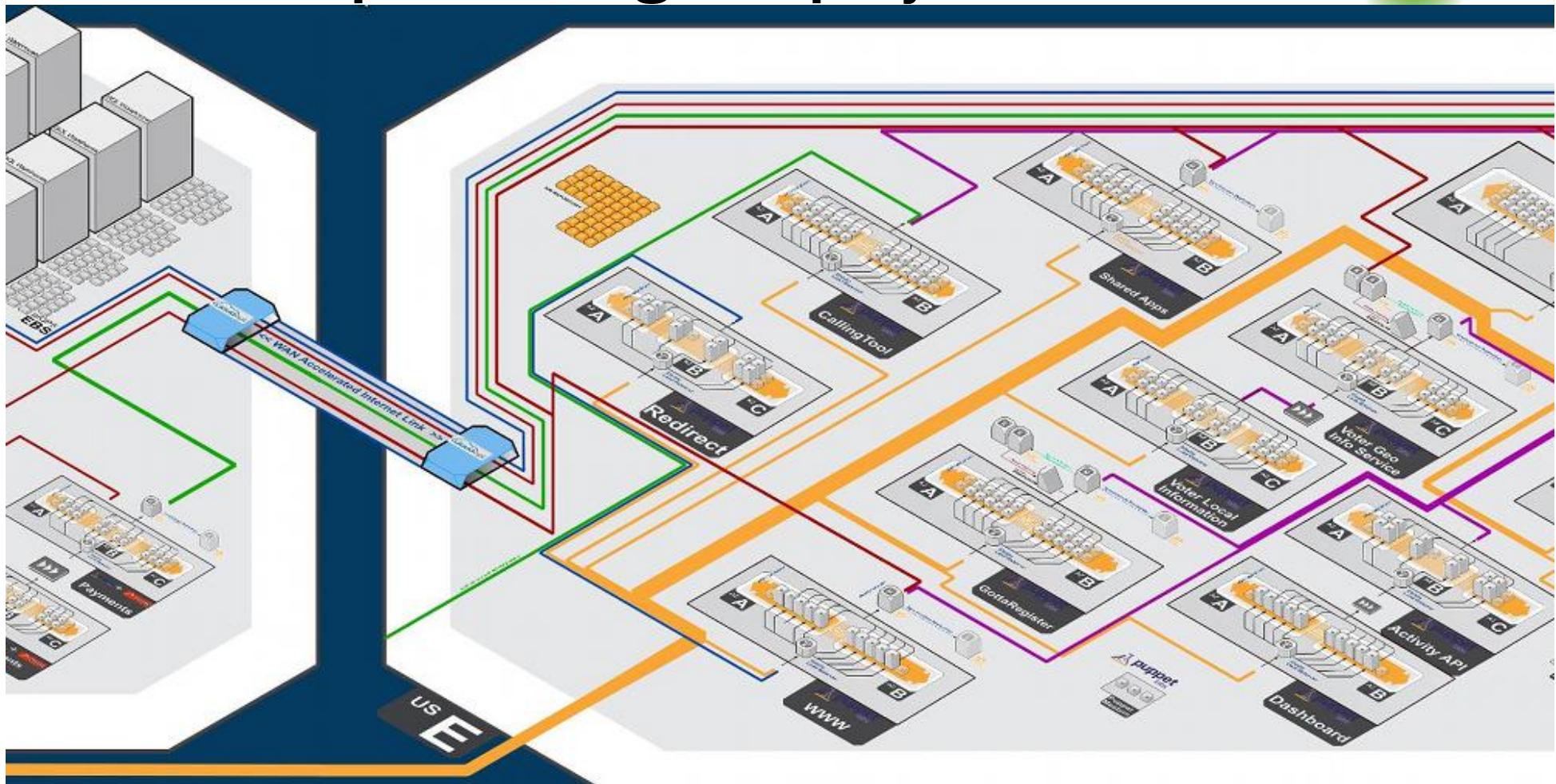
# IaaS is more than virtual machines

---



- Today's IaaS provides wide range of basic services used to build web/enterprise apps in pay-per-use
  - Load balancer, DNS, object storage, CDN, email, queue, SQL/NoSQL database, big data analytics, payment, ...
- Building and operating those services “properly” was often a huge cost for small/mid enterprises
- In IaaS, infrastructure operation is more or less to figure out a “right” combination of given services
  - Less need to worry about fault tolerance of those services

# An Example of Large Deployment in AWS



- [aws.amazon.com/solutions/case-studies/obama/](http://aws.amazon.com/solutions/case-studies/obama/), awsofa.info
  - 200 apps, 2k VMs, 180TB, 10k req/sec
  - Very short lifetime
  - Use IaaS services as much as possible (queue, LB, DB, NoSQL, Analytics, ...)

# Example of PaaS: Google App Engine



- “Platform as a Service” provides a solution stack
- Consumers “deploy” an application (e.g., uploading a war file) – do not touch VM/OS
- Platform manages scaling/backup/security
- Let’s look into Google App Engine





# 1. Create an account. (5 min) GAE offers a large amount of quota for free

java - Eclipse

```
buf.append("&q=" + q.replaceAll("(\\s)+", "+"));
buf.append("&num=30");

return buf.toString();
}

public String query(String input) {

    try{
        URL url = new URL(getQueryUrl(input));

        URLFetchService urlFetchService = URLFetchServiceFactory.getURLFetchService();
        Future<HTTPResponse> response = urlFetchService.fetchAsync(url);

        String content = new String(response.get().getContent());
        QueryResultParser parser = new QueryResultParser(content);

        // Set<Publication> citingPapers = new TreeSet<Publication>(Collections.reverseOrder());

        StringBuffer htmlBuf = new StringBuffer();
        while(parser.hasNext()){
            Publication publication = parser.next();

            htmlBuf.append(publication.toString()).append("<br/>");
        }

        // Escape data from the client to avoid cross-site script vulnerabilities.
        // input = escapeHtml(input);
    }
}
```

Problems @ Javadoc Declaration Console Search Tasks Progress Development Mode

GAERelatedWorkFinder - Deploy to App Engine

Will check again in 4 seconds.  
Will check again in 8 seconds.  
Will check again in 16 seconds.  
Will check again in 32 seconds.  
Closing update: new version is ready to start serving.  
Uploading index definitions.  
Deployment completed successfully

## 2. Write an application using GAE's framework

**Web Application Starter Project - Windows Internet Explorer**

http://relatedworkfinder.appspot.com/

File Edit View Favorites Tools Help

Favorites Suggested Sites Free Hotmail Web Slice Gallery

Web Application Starter Project

Find your related works

cloud computing Search

Search Results References References of References Citings Citings of Citings

Above the clouds: A berkeley view of cloud computing [All 81 versions] cited by 478  
M Armbrust, A Fox, R Griffith, AD Joseph, RH ... - ... , University of California ..., 2009 - Citeseer  
Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia (Comments should be addressed to abovetheclouds@cs.berkeley.edu) ... UC Berkeley Reliable ...

Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities [All 26 versions] cited by 184  
R Buyya, CS Yeo, S Venugopal - ieeexplore.ieee.org  
Abstract This keynote paper: presents a 21st century vision of comp vision of computing utilities; defines Cloud computing and provides t technologies ...

The eucalyptus open-source cloud-computing system [All 27 version]  
D Nurmi, R Wolski, C Grzegorczyk, G ... - ... on Cluster Computing  
Abstract Cloud computing systems fundamentally provide ac- cess to interfaces similar in spirit to exist- ing grid and HPC resource manag new ...

Cloud computing and grid computing 360-degree compared [All 9 ver  
I Foster, Y Zhao, I Raicu, S Lu - ArXiv e-prints, 2008 - adsabs.harvard.edu  
Cloud Computing has become another buzzword after Web 2.0. How there seems to be no consensus on what a Cloud is. On the other ha

Cloud computing and emerging IT platforms: Vision, hype, and reality  
R Buyya, CS Yeo, S Venugopal, J Broberg, I ... - ... Generation Con With the significant advances in Information and Communications Te perceived vision that computing will one day be the 5th utility (after v other ...

Computing in the clouds [All 3 versions] cited by 89  
A Weiss - NetWorker, 2007 - portal.acm.org

**3. Deploy your application on GAE!**

**Scale up/down, load balancing, replication, database management, ... many services are provided by GAE**

Internet 100% 100%

Dashboard - Related Work Finder - Windows Internet Explorer

File Edit View Favorites Tools Help

Favorites Suggested Sites Free Hotmail Web Slice Gallery

Dashboard - Related Work Finder

Google app engine Application: relatedworkfinder Version: 1 shu.wada@gmail.com | My Account | Help | Sign out << My Applications

Main

- [Dashboard](#)
- [Quota Details](#)
- [Logs](#)
- [Cron Jobs](#)
- [Task Queues](#)
- [Blacklist](#)

Data

- [Datastore Indexes](#)
- [Datastore Viewer](#)
- [Datastore Statistics](#)
- [Blob Viewer](#)

Administration

- [Application Settings](#)
- [Permissions](#)
- [Versions](#)
- [Admin Logs](#)

Billing

- [Billing Settings](#)
- [Billing History](#)

Resources

Charts [?](#)

Requests/Second [?](#) 6 hrs 12 hrs 24 hrs 2 days 4 days 7 days 14 days 30 days

Billing Status: Free - [Settings](#) Quotas reset every 24 hours. Next reset: 6 hrs [?](#)

Resource	Usage
CPU Time	0%
Outgoing Bandwidth	0%
Incoming Bandwidth	0%
Total Stored Data	0%
Recipients Emailed	0%

Current Load [?](#)

URI	Requests last 18 hrs	Avg CPU (API) last hr
/gaerelatedworkfinder/relatedworkfinder	9	40 (0)
/gaerelatedworkfinder/gaerelatedworkfinder	5	0 (0)

https://www.google.com/accounts/ManageAccount

Internet 100% [?](#)

**4. Check your resource usage (CPU, storage, # of API calls, ...)**

**Pay only when usage exceeds the free quota**

---

Under the covers of

# “REAL” CLOUD ARCHITECTURE

---

Based on “Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers’ Perspective”, CIDR 2011

# CLOUD ARCHITECTURE OVERVIEW AND NOSQL

# Principle of (Large) Cloud Platform

---



- Cloud = “big computing resource” on top of large number of commodity servers
  - Commodity servers are cheap but not reliable
  - Many “cloud” technologies are designed to achieve high reliability by leveraging massive redundancy
    - Advancement in P2P research in 2000s made a significant contribution to the foundation of cloud

# Computing and Storage

---

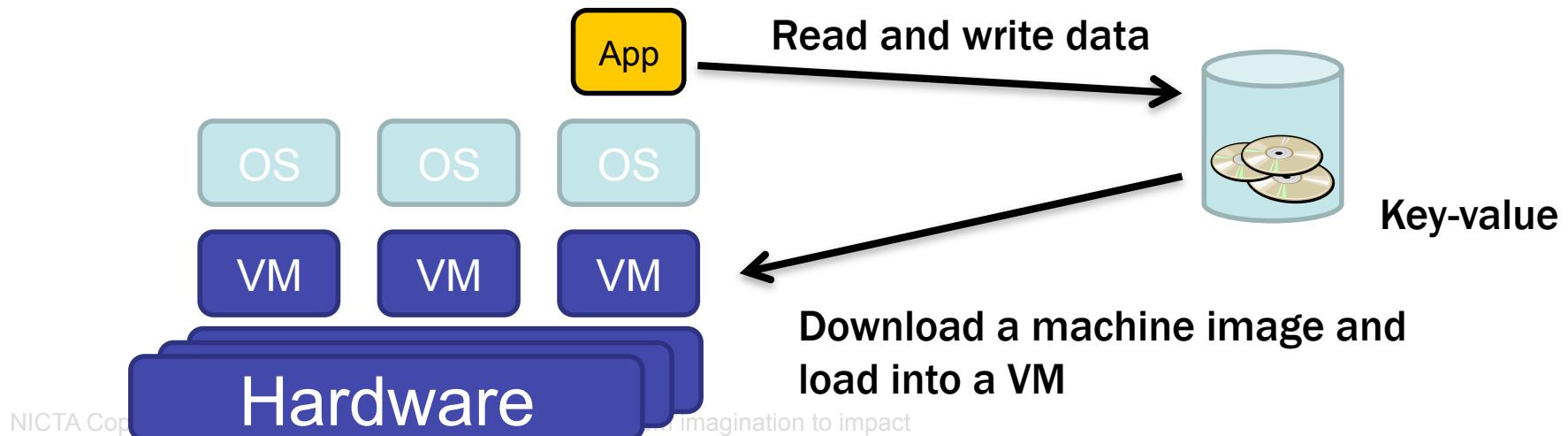


- CPU virtualization enables the sharing CPU resource
  - Relatively easy part since each virtual machine is stateless
- Storage/database technology in cloud has been evolving quickly
  - Today's cloud offer various data storage / database / content-distribution network / distributed cache
  - Common to mix them in one system
  - Critical to understand their pros and cons
- Network virtualization is hot now

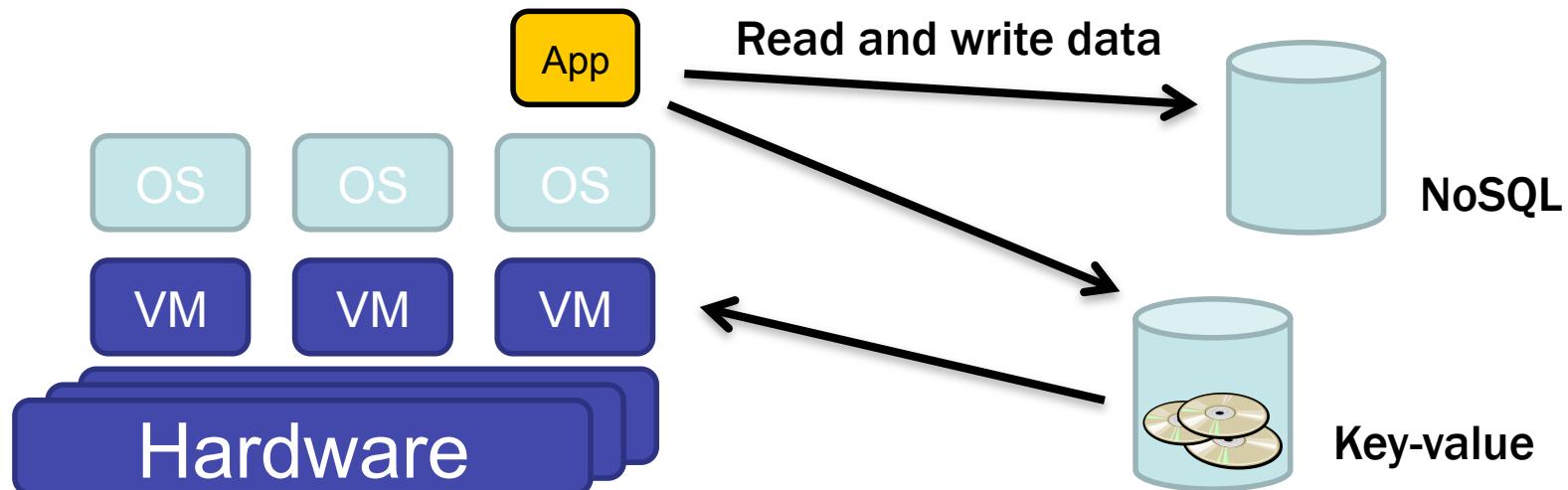
~2008



- “Volatile” VMs and highly available key-value storage
- Each VM can use local non-persistent disks
  - On each boot a VM may “jump” to different physical server
  - On a server failure, a VM restart on a different server
- Replication and P2P (e.g., distributed hash) enables reliable key-value storage



- Vendors provided NoSQL (Not-only SQL) databases as “official” cloud storage
- As cloud computing was getting popular, “NoSQL” became a hot buzzword
  - Many products had emerged



# Why no distributed block storage?

---



- Storage Area Network (SAN) was expensive!
  - Disks and Fibre Channel were not cheap
- Cloud = massive scale and throughput; however, making disk fast and reliable is hard
  - Scaling up is very expensive
  - Scaling out (RAID) is not cheap

# Why no block storage? What's NoSQL?



- Designed to achieve high scalability and high availability for *essential* functionalities, trading off for some features of traditional DBMS
  - No relational data model: key-value or extension of key-value
  - Limited query languages: get/put by key, no or limited join (on same node), range query, ...
  - No or limited transaction
  - Weak consistency: to achieve high availability and throughput on (possibly less reliable) distributed servers
- People wondered “how to build systems on NoSQL?”

# Eventual Consistency

---



- A model originally proposed for disconnected operation (e.g., mobile computing)
- Different nodes keep replicas and each update is “eventually” propagated to each replica
  - DNS is the most well-known system implementing eventual consistency
- Usual definition is *counterfactual*: “once updating ceases, and the system stabilizes, then after a long enough period, all replicas will have the same value”

# Consistency from the Consumer's Perspective

NICTA

- Work done in “Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers’ Perspective”, CIDR 2011
- Investigate consistency models provided by commercial NoSQLs in cloud
  - If weak, which extra properties supported? How often and in what circumstances is inconsistency (stale values) observed?
  - Any differences between what is observed and what is announced from the vendor?

# Platforms we observed

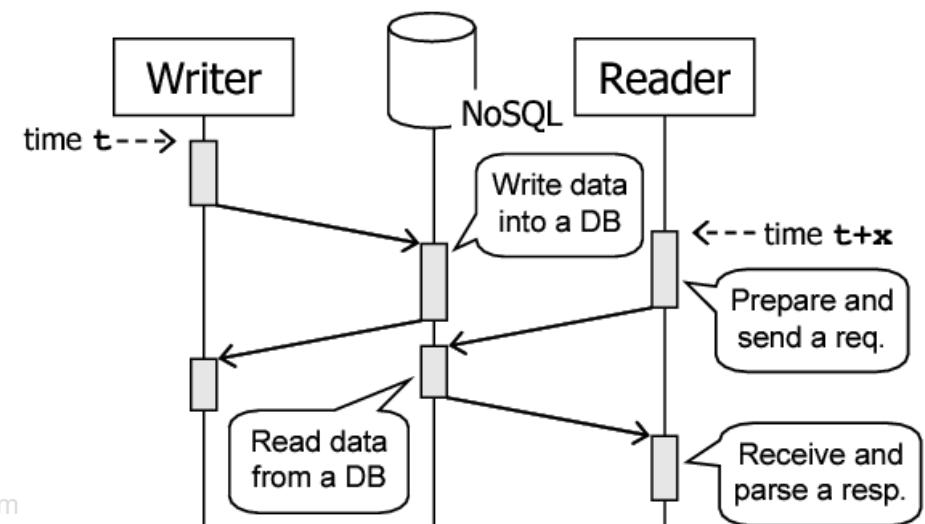
---

- A variety of commercial cloud NoSQLs that are offered as storage service
  - Amazon S3
    - Two options: Regular and Reduced redundancy (durability)
  - Amazon SimpleDB
    - Two options: Consistent Reads and Eventual Consistent Reads
    - (DynamoDB was introduced in 2012)
  - Google App Engine datastore
    - Two options: Strong and Eventual Consistent Reads (added “High Replication” in 2011)
  - Windows Azure Table and Blob
    - No option available in data consistency

# Frequency of Observing Stale Data



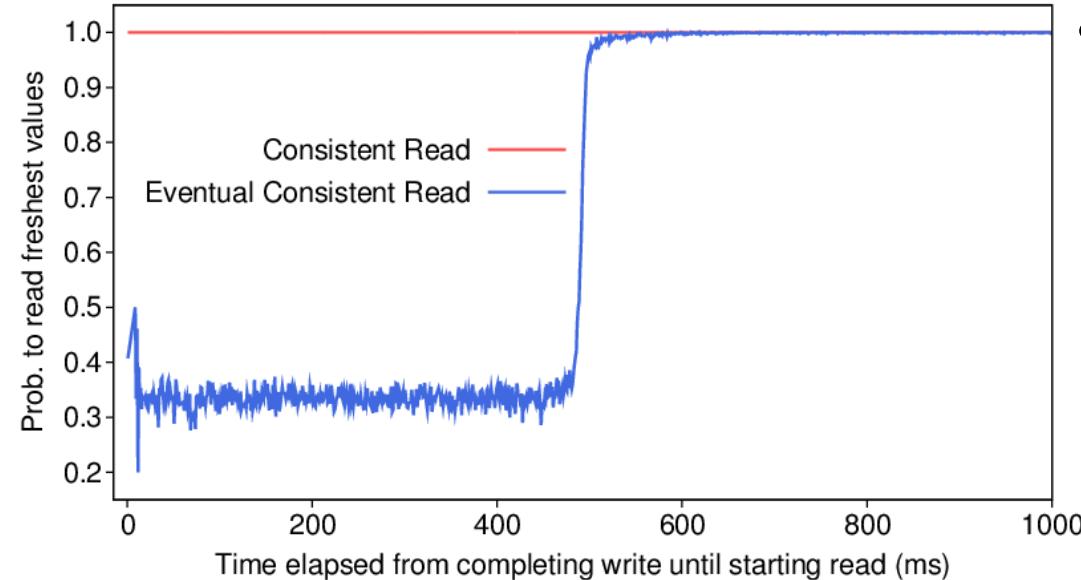
- **Experimental Setup**
  - A thread (or process) writes timestamp in storage
  - Another thread (or process) keeps reading from the storage
    - 100 times/sec
    - Check if the data is stale by comparing value seen to the most recent value written
    - Plot against time since most recent write occurred



# SimpleDB: Read and Write from a Single Thread

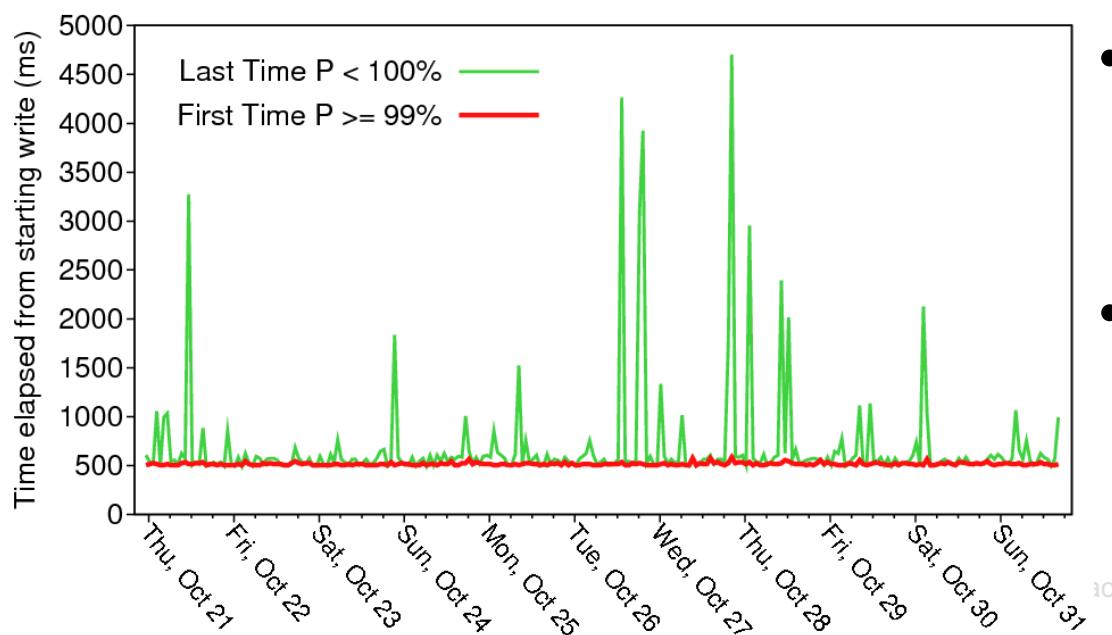


NICTA



- With eventual consistent read, 33% of chance to read freshest values within 500ms

- Perhaps one master and two other replicas. Read takes value randomly from one of these?



- First time for eventual consistent read to reach 99% “fresh” is stable 500ms
- Outlier cases of stale read after 500ms, but no regular daily or weekly variation observed

# Stale Data in Other Cloud NoSQLs



Cloud NoSQL and Accessing Source	What Observed
SimpleDB (access from one thread, two threads, two processes, two VMs or two regions)	<u>Accessing source has no affect on the observable consistency.</u> Eventual consistent reads have 33% chance to see stale value, till 500ms after write.
S3 (with five access configurations)	No stale data was observed in ~4M reads/config. <u>Providing better consistency than SLA describes.</u>
GAE datastore (access from a single app or two apps)	Eventual consistent reads from different apps have very low (3.3E <sup>-4</sup> %) chance to observe values older than previous reads. Other reads never saw stale data.
Azure Storages (with five access configurations)	No stale data observed. Matches SLA described by the vendor (all reads are consistent).

# Additional Properties: Read-Your-Writes?



- **Read-your-writes:** a read always sees a value at least as fresh as the latest write from the same thread/session
  - Once you write x, you will read x
- **SimpleDB with eventual consistent read:** does not have this property
- **GAE with eventual consistent read:** may have this property
  - No counterexample observed in ~3.7M reads over two weeks

# Additional Properties: Monotonic Reads?



NICTA

- **Monotonic Reads:** Each read sees a value at least as fresh as that seen in any previous read from the same thread/session
  - Write x then y. Once you read y, you never see x.
- Our experiment: check for a fresh read followed by a stale one
- SimpleDB: Monotonic read consistency is not supported
  - In staleness, two successive eventual consistent reads are almost independent
- GAE with eventual consistent read: not supported
  - 3.3E-4 % chance to read values older than previous reads

	2 <sup>nd</sup> Stale	2 <sup>nd</sup> Fresh
1 <sup>st</sup> Stale	39.94% (~1.9M)	21.08% (~1.0M)
1 <sup>st</sup> Fresh	23.36% (~1.1M)	15.63% (~0.7M)

# Additional Properties: Monotonic Writes?

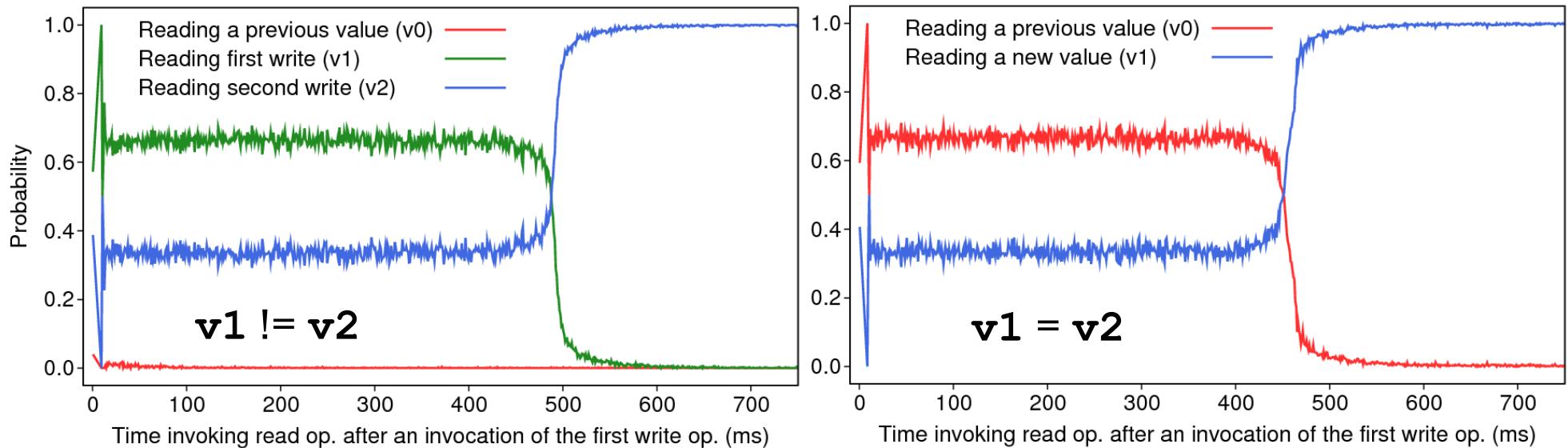


- **Monotonic Writes:** Each write is completed in a replica after previous writes have been completed
  - Write x then y. you may read y then read x sometime
- Programming is “notoriously hard” if monotonic write consistency is missing
  - W. Vogels. Eventually consistent. Commun. ACM, 52(1), 2009.

# SimpleDB's Eventual Consistent Read: Monotonic Write



- A data has value  $v_0$  before each run. Writing value  $v_1$  and then  $v_2$  there, then read it repeatedly



- When  $v_1 \neq v_2$ , writing  $v_2$  “pushes”  $v_1$  to replicas immediately (previous value  $v_0$  is not observed)
- When  $v_1 = v_2$ , second write does not push ( $v_0$  is observed)
  - Same as the “only writing one value” case

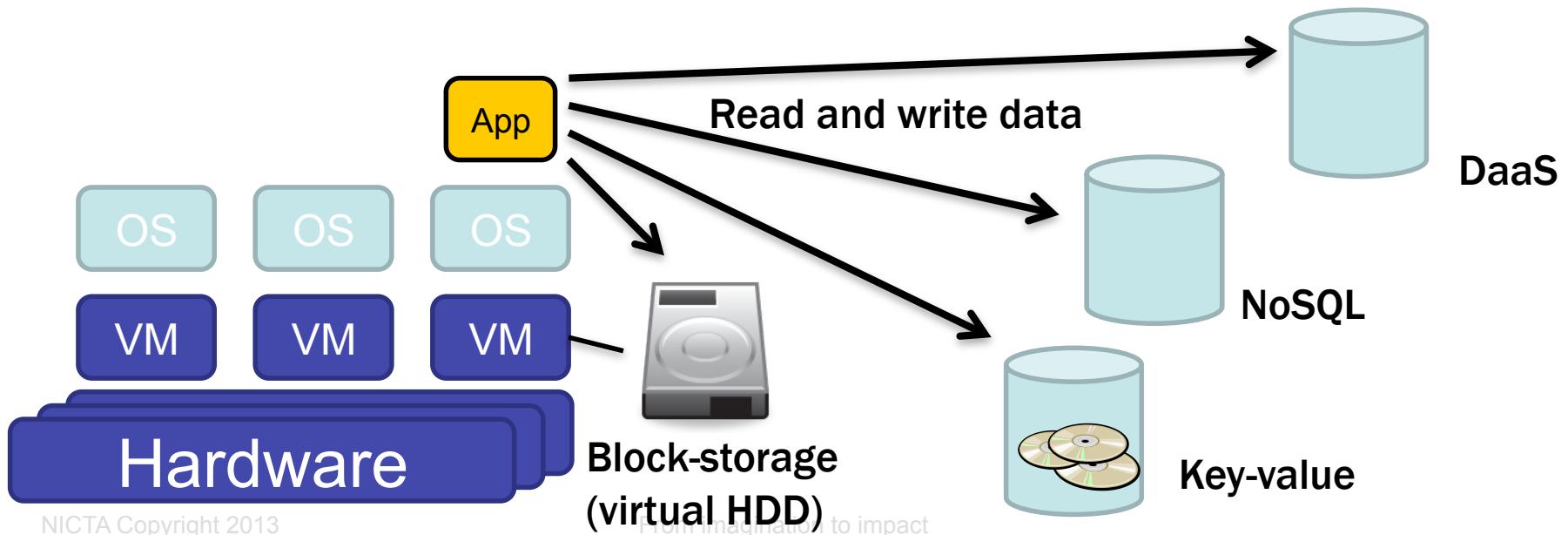
# What Consumers Can Observe

---

- SimpleDB platform showed frequent inconsistency
- It offers option for consistent read. No extra costs for the consumer were observed from our experiments
  - At least under the scale of our experiments (few KB stored in a domain and ~2,500 rps)
- Some platforms gave (mostly) better consistency than they promise
  - Consistency almost always (5-nines or better)
  - Perhaps consistency violated only when network or node failure occurs during execution of the operation

# 2010 to 2011

- Vendors started providing virtual persistent block storage in cloud
  - on Storage Area Networks (SANs)
- Also started providing RDBMS services in cloud
  - Database-as-a-service
  - RDBMS pre-installed VMs and managed by vendors



# Trends in NoSQL after 2011

---



- “NoSQL” is accepted as a new class of storage
  - High throughput, less system administration, easy to build fault tolerance over geo-replication, schema less and easy to write client code, ...
- Popular NoSQL implementations are everywhere
  - DynamoDB, MongoDB, CouchDB, Redis, ...
  - But no sign of “common NoSQL model/API” yet
- Many NoSQL products offer consistent view
  - Some offer tunable consistency
- Research effort to “revive missing features” on NoSQL

---

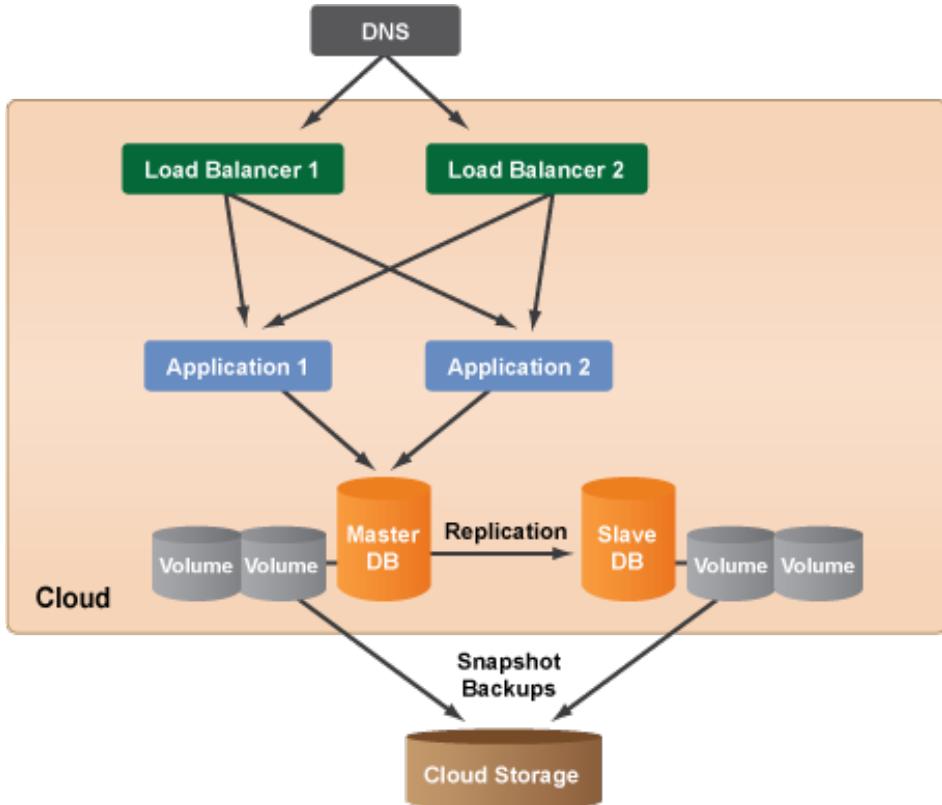
# APPLICATION ARCHITECTURE AND OPERATIONS IN CLOUD

# Design for Scale

---

- **Scale horizontally (out/in)**
  - Adding/removing nodes to/from a system
  - Architecture could be complex
  - Highly available
  - Good for stateless layers (e.g., web servers and computing array)
- **Scale vertically (up/down)**
  - Adding/removing resource to/from a node
  - Easy to implement but potentially expensive
  - Hard to avoid downtime on scale, less availability
  - Good for stateful layers (e.g., relational database)
- **For small-mid applications, scale up/down is not a bad choice!**

# 3-Tier Architecture for Scale



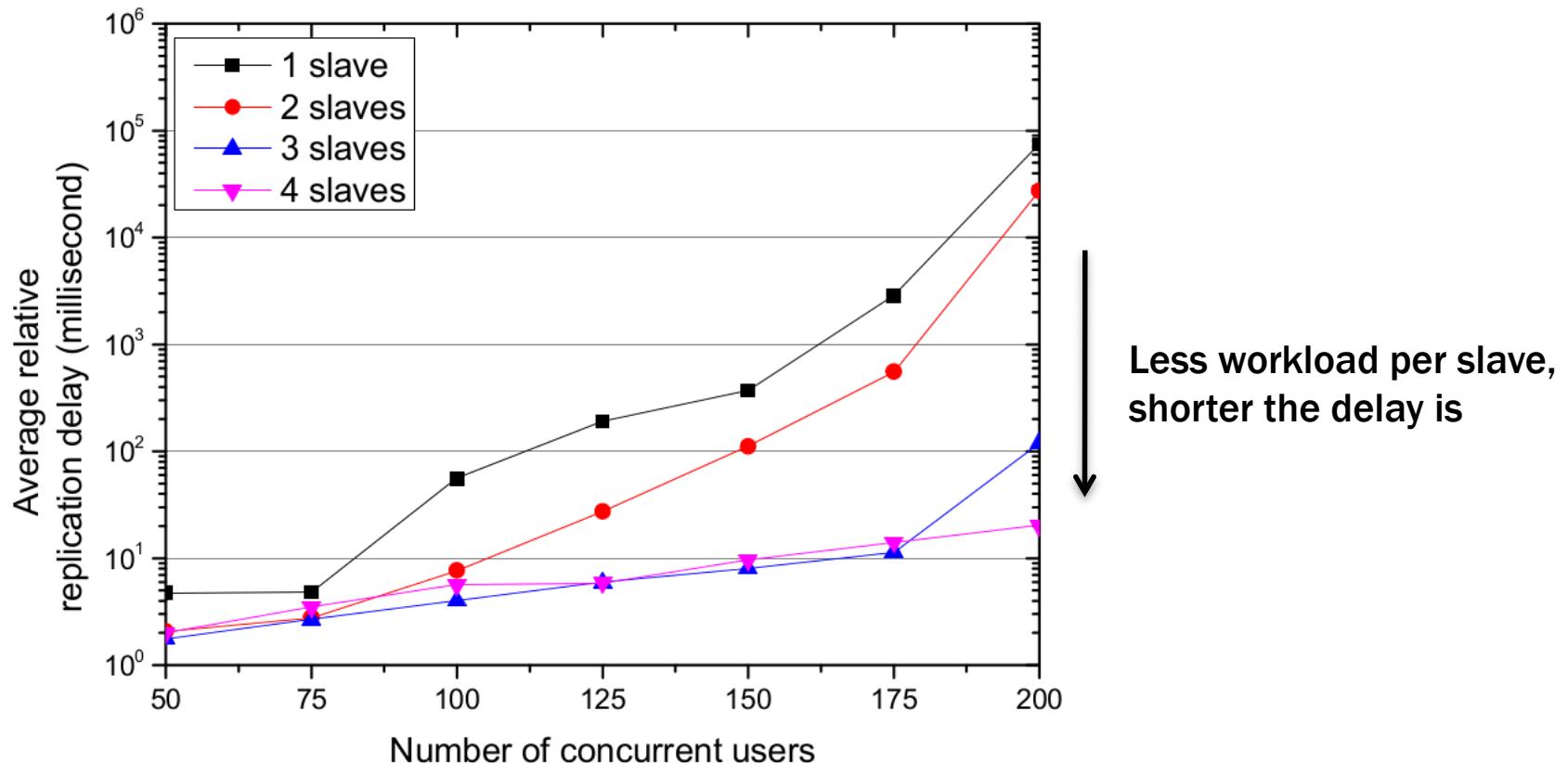
[support.rightscale.com/12-Guides/  
EC2\\_Best\\_Practices/EC2\\_Site\\_Architecture\\_Diagrams](http://support.rightscale.com/12-Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams)

- **Dynamic DNS**
  - Distribute traffic among public-facing endpoints
- **Load-balancer**
  - Can handle large traffic
  - Easy to scale out
- **Application layer**
  - Stateless and easy to scale out
  - Cache efficiency, session stickyness, ...
- **Persistent layer**
  - Stateful and hard to scale out
  - Master-slave for scaling and fault tolerance

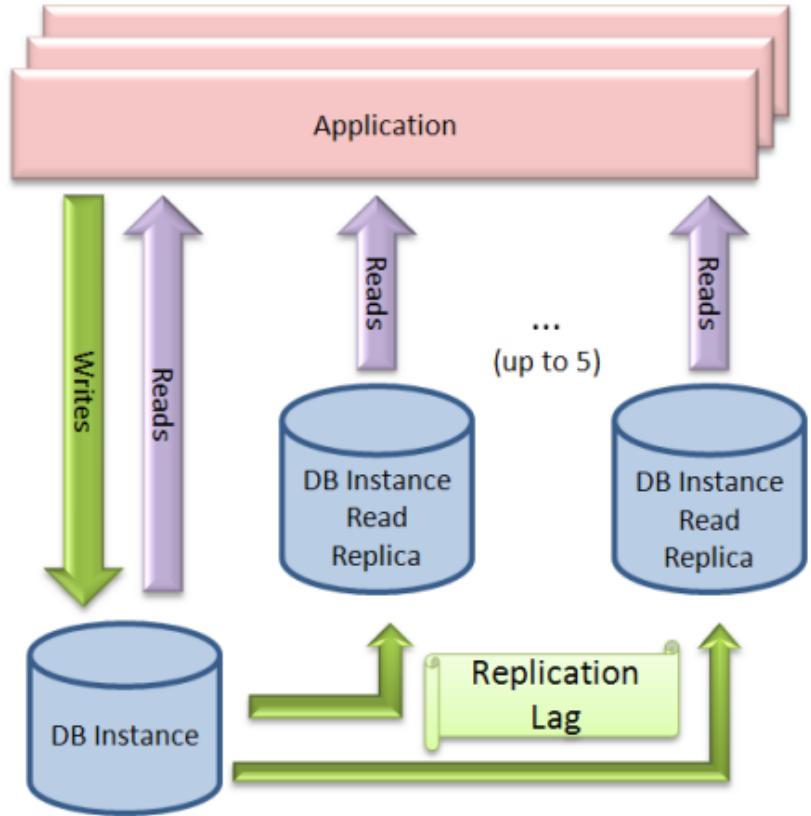
# Master-Slave Architecture (Persistent Layer)



- Create a slave(s) for two purposes: scaling and fault tolerance
- Asynchronous replication is used often → replication delay between a master and the slave



# Read-Replica

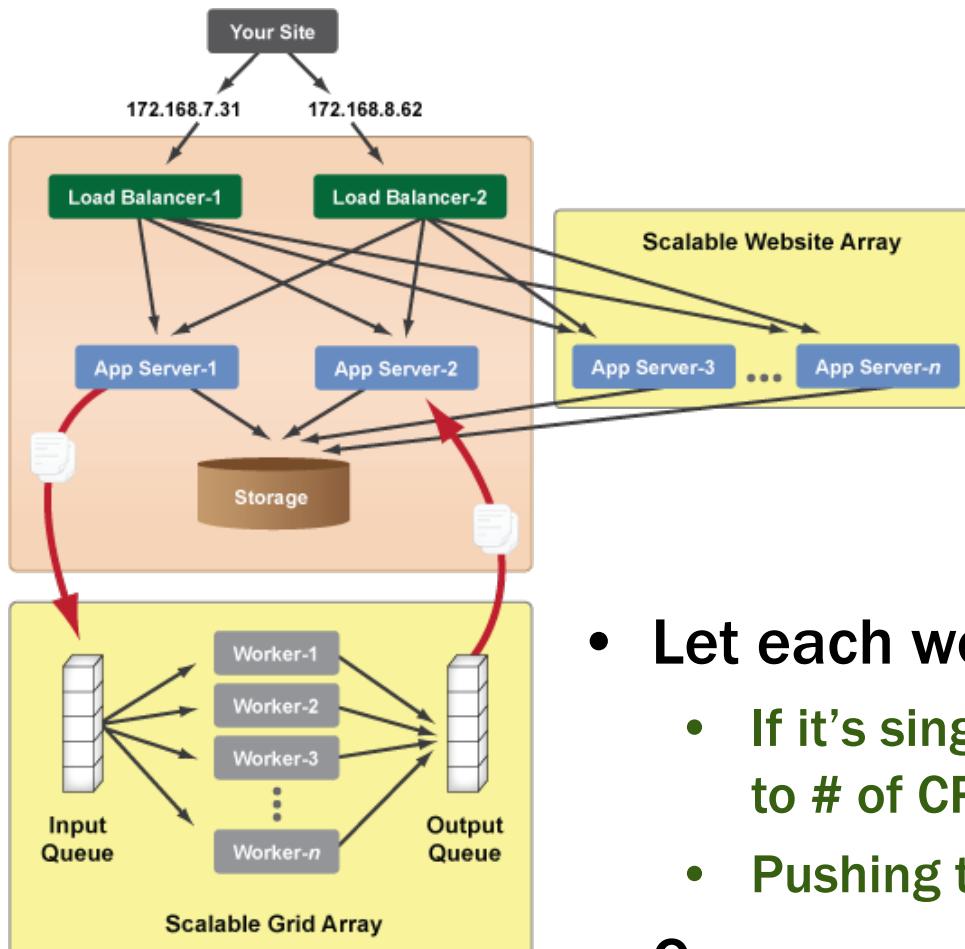


[aws.typepad.com/aws/2010/10/amazon-rds-announcing-read-replicas.html](http://aws.typepad.com/aws/2010/10/amazon-rds-announcing-read-replicas.html)

- Using a slave as a read-only copy is a common technique
- Achieving FT and scaling
  - Typical read-write ratio is 90:10
  - Due to replication delay, it's good to read only from read replicas
  - Only one slave and forward 90% transactions to it does not scale out
  - Using a master and the slave over 50% of capacity → Two “single point of failures”



NICTA



- Long running tasks should be executed asynchronously
  - Holding HTTP connections wastes resource
- Easy to scale in/out
  - Adjust the # of workers (VMs) based on the queue length
- Let each worker adjust the concurrency
  - If it's single threaded, each worker can process up to # of CPUs
  - Pushing tasks poses the risk of node crash
- Concerns
  - Timing of scaling out/in, restarting jobs upon a node failure or scaling in, prevent nodes from popping the same request, task priority, ...

[support.rightscale.com/12-Guides/  
EC2\\_Best\\_Practices/  
EC2\\_Site\\_Architecture\\_Diagrams](http://support.rightscale.com/12-Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams)

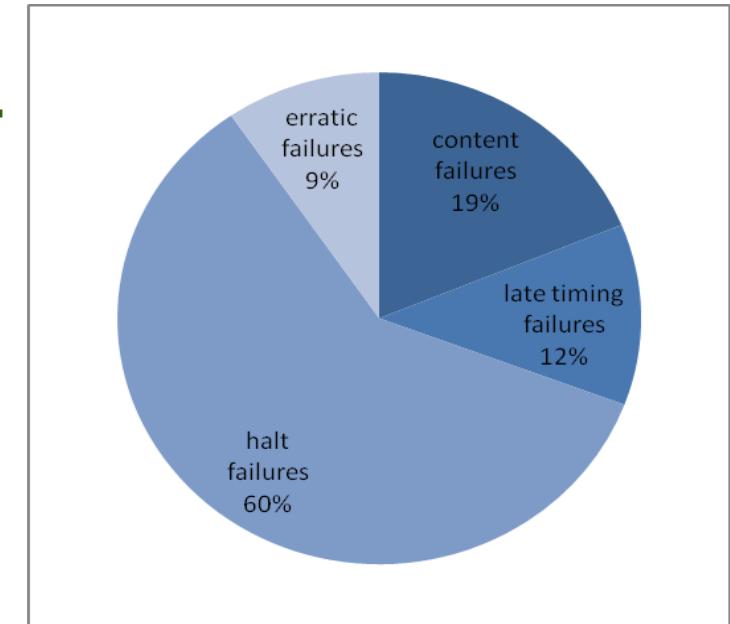
# Reliability of Cloud

---

- (Public) cloud computing is a “best-effort” service
- E.g., Amazon’s SLA on availability is 99.95%
  - AWS will use commercially reasonable efforts to make Amazon EC2 available with an Annual Uptime Percentage of at least 99.95%
  - “Annual Uptime Percentage” is calculated by subtracting from 100% the percentage of 5 minute periods during the Service Year in which Amazon EC2 was in the state of “Region Unavailable.”
  - “Region Unavailable” and “Region Unavailability” means that more than one Availability Zone in which you are running an instance, within the same Region, is “Unavailable” to you.
- Availability of each VM could be much lower than 99.95%

# You have to prepare for failures

- Failures are common in cloud
  - No responding VM due to an underlying hardware (network) failure
  - Wrong result from an API call
  - Takes too long to process an request (not a “failure”)
  - An API call fails
  - A resource gets ‘stuck’ in an error state
  - A data center wide failure (e.g., network connection)
  - Reboot request



“API Issues: an Empirical Study and Impact”, QoS 2013

# Design for Failure



- Cloud providers explicitly say they do not guarantee 100% availability
  - They provide redundancy instead
- You must architect your system to survive random failures

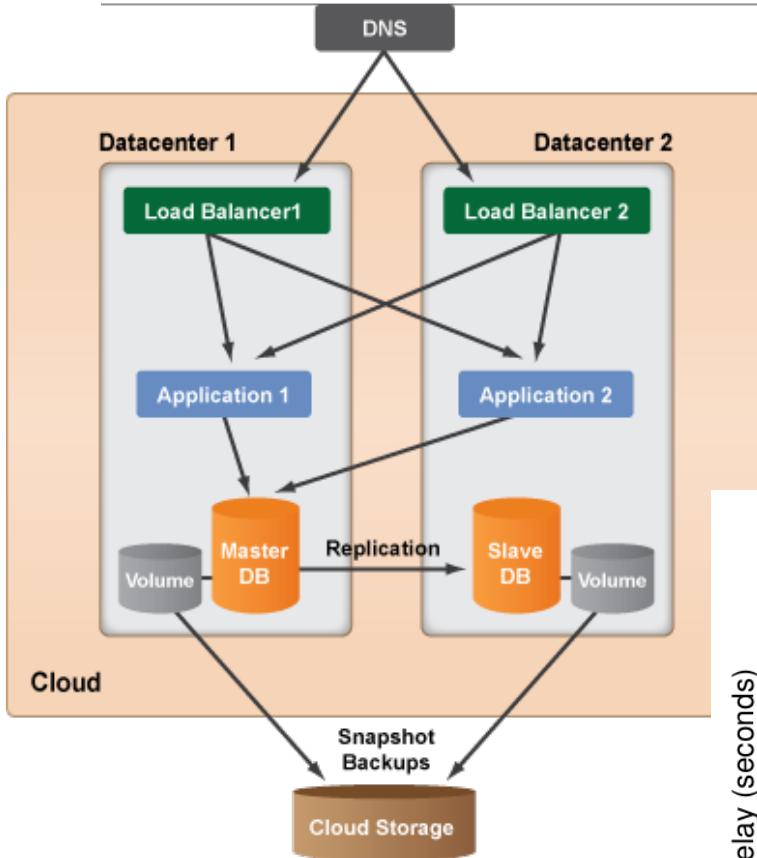
<http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html>

- Chaos monkey: test your system against random failures
  - “The best way to avoid failure is to fail constantly” by Netflix
  - Randomly shutdown virtual machines, services, ...

# Geographically-distributed architecture

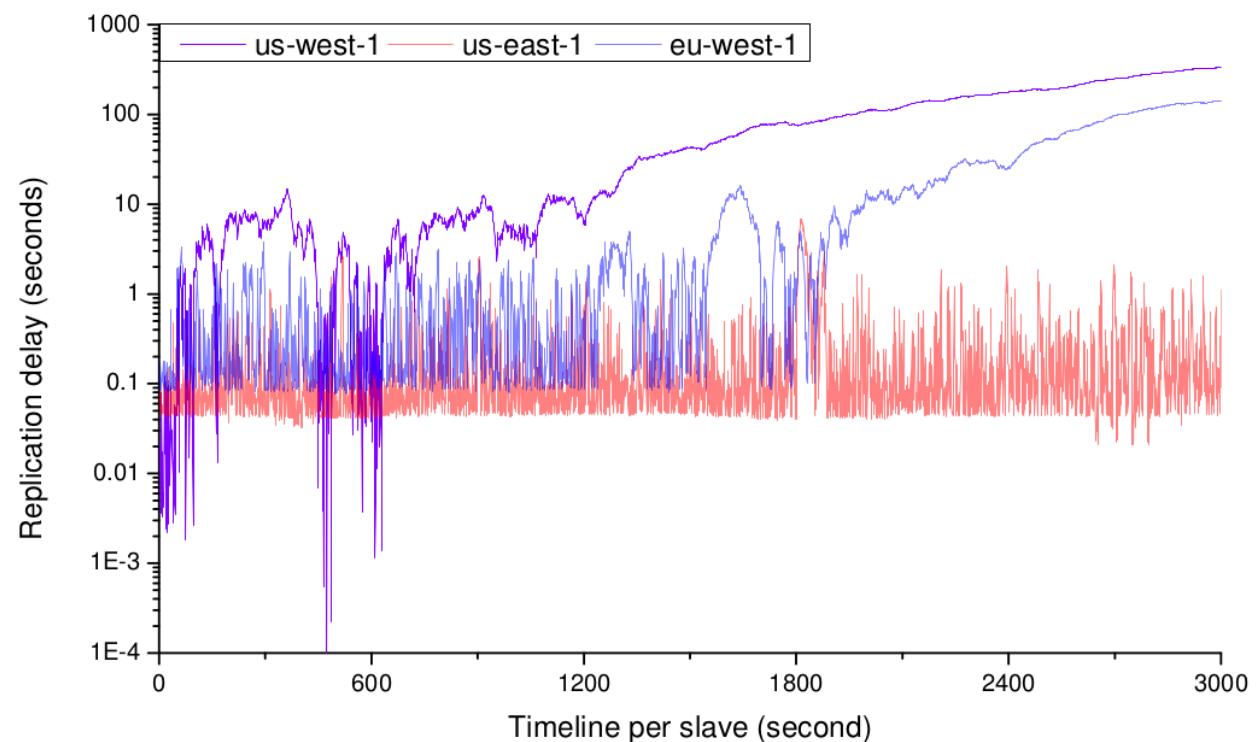


NICTA



[support.rightscale.com/12-Guides/  
EC2\\_Best\\_Practices/  
EC2\\_Site\\_Architecture\\_Diagrams](http://support.rightscale.com/12-Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams)

- Cloud allows for using geographically dispersed data centers
  - > 50km away, on different utility infrastructure (e.g., power, water, road, ...)
  - Traditionally very expensive solution
  - The speed of light is your enemy!



# Failure Handling at Yuruware

---



- Chaos monkey is not only for random failures
  - Strange errors occur everywhere: cloud, os, app, ...
- Easiest error handling = discard and get new one
- What we're doing
  - Worker VM kills itself if detecting a fatal app error
  - Worker VM kills itself if not executing jobs or not receiving HB from management server for an hour
  - The mgmt server kills workers if not responding
  - Let processes/VMs clean up the mess and restart from start
- Key design: error detection, autonomous decision making, loose coupling and async process
- “micro-rebooting” in cloud makes your app robust

# Configuration Management



- Config management system is a great help
  - Puppet, Chef, ...

```
node basenode {  
    $bolt_opt_dir = "/opt/ybolt.net"  
    include env::swapfile  
    include env::hostname  
    class { logwatch :  
        logwatch_mail_to => $ops_email_addr  
    }  
}  
node 'prod.ybolt.net' inherits boltserver {  
    mysql::db { "$mysql_bolt_db" }  
    class { mysql_s3_backup : s3_bucket => "..." }  
}
```

- Make all environments consist
- Restart an environment quickly
  - Micro-rebooting, moving from local to cloud, ...