IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

# CLOUD COMPUTING

## SERVICE LEVEL AGREEMENT (SLA)


**PROF. SOUMYA K. GHOSH**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IIT KHARAGPUR

# What is Service Level Agreement?

- A formal contract between a Service Provider (SP) and a Service Consumer (SC)
- SLA: foundation of the consumer's trust in the provider
- Purpose : to define a formal basis for performance and availability the SP guarantees to deliver
- SLA contains Service Level Objectives (SLOs)
  - Objectively measurable conditions for the service
  - SLA & SLO: basis of selection of cloud provider

# SLA Contents

- A set of services which the provider will deliver
- A complete, specific definition of each service
- The responsibilities of the provider and the consumer
- A set of metrics to measure whether the provider is offering the services as guaranteed
- An auditing mechanism to monitor the services
- The remedies available to the consumer and the provider if the terms are not satisfied
- How the SLA will change over time

# Web Service SLA

- WS-Agreement
  - XML-based language and protocol for negotiating, establishing, and managing service agreements at runtime
  - Specify the nature of agreement template
  - Facilitates in discovering compatible providers
  - Interaction : request-response
  - SLA violation : dynamically managed and verified
- WSLA (Web Service Level Agreement Framework)
  - Formal XML-schema based language to express SLA and a runtime interpreter
  - Measure and monitor QoS parameters and report violations
  - Lack of formal definitions for semantics of metrics

# Difference between Cloud SLA and Web Service SLA

- QoS Parameters :
  - Traditional Web Service : response time, SLA violation rate for reliability, availability, cost of service, etc.
  - Cloud computing : QoS related to security, privacy, trust, management, etc.
- Automation :
  - Traditional Web Service : SLA negotiation, provisioning, service delivery, monitoring are not automated.
  - Cloud computing : SLA automation is required for highly dynamic and scalable service consumption
- Resource Allocation :
  - Traditional Web Service : *UDDI (Universal Description Discovery and Integration)* for advertising and discovering between web services
  - Cloud computing : resources are allocated and distributed globally without any central directory

# Types of SLA

- Present market place features two types of SLAs :
  - Off-the-shelf SLA or non-negotiable SLA or Direct SLA
    - Non-conducive for mission-critical data or applications
    - Provider creates the SLA template and define all criteria viz. contract period, billing, response time, availability, etc.
    - *Followed by the present day state-of-the-art clouds*.
  - Negotiable SLA
    - Negotiation via external agent
    - Negotiation via multiple external agents

# Service Level Objectives (SLOs)

- Objectively measurable conditions for the service
- Encompasses multiple QoS parameters viz. availability, serviceability, billing, penalties, throughput, response time, or quality
- Example :
  - "***Availability*** of a service X is 99.9%"
  - "***Response time*** of a database query Q is between 3 to 5 seconds"
  - "***Throughput*** of a server S at peak load time is 0.875"

# Service Level Management

- Monitoring and measuring performance of services based on SLOs
- Provider perspective :
  - Make decisions based on business objectives and technical realties
- Consumer perspective :
  - Decisions about how to use cloud services

# Considerations for SLA

- **Business Level Objectives**: Consumers should know *why* they are using cloud services before they decide *how* to use cloud computing.
- **Responsibilities of the Provider and Consumer**: The balance of responsibilities between providers and consumers will vary according to the type of service.
- **Business Continuity and Disaster Recovery**: Consumers should ensure their cloud providers have adequate protection in case of a disaster.
- **System Redundancy**: Many cloud providers deliver their services via massively redundant systems. Those systems are designed so that even if hard drives or network connections or servers fail, consumers will not experience any outages.

# Considerations for SLA *(contd…)*

- **Maintenance**: Maintenance of cloud infrastructure affects any kind of cloud offerings (applicable to both software and hardware)
- **Location of Data**: If a cloud service provider promises to enforce data location regulations, the consumer must be able to audit the provider to prove that regulations are being followed.
- **Seizure of Data**: If law enforcement targets the data and applications associated with a particular consumer, the multi-tenant nature of cloud computing makes it likely that other consumers will be affected. Therefore, the consumer should consider using a third-party to keep backups of their data
- **Failure of the Provider**: Consumers should consider the financial health of their provider and make contingency plans. The provider's policies of handling data and applications of a consumer whose account is delinquent or under dispute are to be considered.
- **Jurisdiction**: Consumers should understand the laws that apply to any cloud providers they consider.

# SLA Requirements

- **Security**: Cloud consumer must understand the controls and federation patterns necessary to meet the security requirements. Providers must understand what they should deliver to enable the appropriate controls and federation patterns.
- **Data Encryption**: Details of encryption and access control policies.
- **Privacy**: Isolation of customer data in a multi-tenant environment.
- **Data Retention and Deletion**: Some cloud providers have legal requirements of retaining data even of it has been deleted by the consumer. Hence, they must be able to prove their compliance with these policies.
- **Hardware Erasure and Destruction**: Provider requires to zero out the memory if a consumer powers off the VM or even zero out the platters of a disk, if it is to be disposed or recycled.

# SLA Requirements *(Contd…)*

- **Regulatory Compliance**: If regulations are enforced on data and applications, the providers should be able to prove compliance.
- **Transparency**: For critical data and applications, providers must be proactive in notifying consumers when the terms of the SLA are breached.
- **Certification**: The provider should be responsible in proving the certification of any kind of data or applications and keeping its up-to date.
- **Monitoring**: To eliminate the conflict of interest between the provider and the consumer, a neural third-party organization is the best solution to monitor performance.
- **Auditability**: As the consumers are liable to any breaches that occur, it is vital that they should be able to audit provider's systems and procedures. An SLA should make it clear how and when those audits take place. Because audits are disruptive and expensive, the provider will most likely place limits and charges on them.

# Key Performance Indicators (KPIs)

- Low-level resource metrics
- Multiple *KPIs* are composed, aggregated, or converted to for high-level *SLOs*.
- Example :
  - downtime, uptime, inbytes, outbytes, packet size, etc.
- Possible mapping :
  - ***Availability (A) = 1 – (downtime/uptime)***

# Industry-defined KPIs

- Monitoring:
  - Natural questions:
    - “who should monitor the performance of the provider?”
    - “does the consumer meet its responsibilities?”
  - Solution: neutral third-party organization to perform monitoring
  - Eliminates conflicts of interest if:
    - Provider reports outage at its sole discretion
    - Consumer is responsible for an outage
- Auditability:
  - Consumer requirement:
    - Is the provider adhering to legal regulations or industry-standard
    - SLA should make it clear how and when to conduct audits

# Metrics for Monitoring and Auditing

- **Throughput** – How quickly the service responds
- **Availability** – Represented as a percentage of uptime for a service in a given observation period.
- **Reliability** – How often the service is available
- **Load balancing** – When elasticity kicks in (new VMs are booted or terminated, for example)
- **Durability** – How likely the data is to be lost
- **Elasticity** – The ability for a given resource to grow infinitely, with limits (the maximum amount of storage or bandwidth, for example) clearly stated
- **Linearity** – How a system performs as the load increases

# Metrics for Monitoring and Auditing *(Contd…)*

- **Agility** – How quickly the provider responds as the consumer's resource load scales up and down
- **Automation** – What percentage of requests to the provider are handled without any human interaction
- **Customer service response times** – How quickly the provider responds to a service request. This refers to the human interactions required when something goes wrong with the on-demand, self-service aspects of the cloud.
- **Service-level violation rate** – Expressed as the mean rate of SLA violation due to infringements of the agreed warranty levels.
- **Transaction time** – Time that has elapsed from when a service is invoked till the completion of the transaction, including the delays.
- **Resolution time** – Time period between detection of a service problem and its resolution.

# SLA Requirements w.r.t. Cloud Delivery Models

| Requirement | Platform as a Service | Infrastructure as a Service | Software as a Service |
|---|---|---|---|
| Data Encryption | ✓ | ✓ | |
| Privacy | ✓ | ✓ | ✓ |
| Data Retention and Deletion | | ✓ | ✓ |
| Hardware Erasure and Destruction | | ✓ | ✓ |
| Regulatory Compliance | ✓ | ✓ | ✓ |
| Transparency | ✓ | ✓ | ✓ |
| Certification | ✓ | ✓ | ✓ |
| Terminology for Key Performance Indicators | | ✓ | ✓ |
| Metrics | ✓ | ✓ | ✓ |
| Auditability | ✓ | ✓ | ✓ |
| Monitoring | ✓ | ✓ | ✓ |
| Machine-Readable SLAs | | ✓ | |



*Source: "Cloud Computing Use Cases White Paper" Version 4.0*

# Example Cloud SLAs

| Cloud Provider | Service | Type of Delivery Model | Service Level Agreement Guarantees |
|---|---|---|---|
| Amazon | EC2 | IaaS | Availability (99.95%) with the following definitions : Service Year : 365 days of the year, Annual Percentage Uptime, Region Unavailability : no external connectivity during a five minute period, Eligible Credit Period, Service Credit |
| | S3 | Storage-as-a-Service | Availability (99.9%) with the following definitions: Error Rate, Monthly Uptime Percentage, Service Credit |
| | SimpleDB | Database-as-a-Service | No specific SLA is defined and the agreement does not guarantee availability |
| Salesforce | CRM | PaaS | No SLA guarantees for the service provided |
| Google | Google App Engine | PaaS | Availability (99.9%) with the following definitions : Error Rate, Error Request, Monthly Uptime Percentage, Scheduled Maintenance, Service Credits, and SLA exclusions |

# Example Cloud SLAs *(contd...)*

| Cloud Provider | Service | Type of Delivery Model | Service Level Agreement Guarantees |
|---|---|---|---|
| Microsoft | Microsoft Azure Compute | IaaS/PaaS | Availability (99.95%) with the following definitions : Monthly Connectivity Uptime Service Level, Monthly Role Instance Uptime Service Level, Service Credits, and SLA exclusions |
| | Microsoft Azure Storage | Storage-as-a-Service | Availability (99.9%) with the following definitions: Error Rate, Monthly Uptime Percentage, Total Storage Transactions, Failed Storage Transactions, Service Credit, and SLA exclusions |
| Zoho suite | Zoho mail, Zoho CRM, Zoho books | SaaS | Allows the user to customize the service level agreement guarantees based on : Resolution Time, Business Hours & Support Plans, and Escalation |

# Example Cloud SLAs *(contd...)*

| Cloud Provider | Service | Type of Cloud Delivery Model | Service Level Agreement Guarantees |
|---|---|---|---|
| Rackspace | Cloud Server | IaaS | Availability regarding the following: Internal Network (100%), Data Center Infrastructure (100%), Load balancers (99.9%)<br>Performance related to service degradation: Server migration, notified 24 hours in advance, and is completed in 3 hours (maximum)<br>Recovery Time: In case of failure, guarantee of restoration/recovery in 1 hour after the problem is identified. |
| Terremark | vCloud Express | IaaS | Monthly Uptime Percentage (100%) with the following definitions: Service Credit, Credit Request and Payment Procedure, and SLA exclusions |

# Example Cloud SLAs *(contd…)*

| Cloud Provider | Service | Type of Cloud Delivery Model | Service Level Agreement Guarantees |
|---|---|---|---|
| Nirvanix | Public, Private, Hybrid Cloud Storage | Storage-as-a-Service | Monthly Availability Percentage (99.9%) with the following definitions: Service Availability, Service Credits, Data Replication Policy, Credit Request Procedure, and SLA Exclusions |

# Limitations

- Service measurement
  - Restricted to uptime percentage
  - Measured by taking the mean of service availability observed over a specific period of time
  - Ignores other parameters like stability, capacity, etc.
- Biasness towards vendors
  - Measurement of parameters are mostly established according to vendor's advantage
- Lack of active monitoring on customer's side
  - Customers are given access to some ticketing systems and are responsible for monitoring the outages.
  - Providers do not provide any access to active data streams or audit trails, nor do they report any outages.

# Limitations *(contd…)*

- Gap between *QoS hype* and *SLA offerings* in reality
- QoS in the areas of *governance*, *reliability*, *availability*, *security*, and *scalability* are not well addressed.
- No formal ways of verifying if the SLA guarantees are complying or not.
- Proper SLA are good for both provider as well as the customer
  - Provider's perspective : Improve upon Cloud infrastructure, fair competition in Cloud market place
  - Customer's perspective : Trust relationship with the provider, choosing appropriate provider for moving respective businesses to Cloud

# Expected SLA Parameters

- Infrastructure-as-a-Service (IaaS):
  - CPU capacity, cache memory size, boot time of standard images, storage, scale up (maximum number of VMs for each user), scale down (minimum number of VMs for each user), On demand availability, scale uptime, scale downtime, auto scaling, maximum number of VMs configured on physical servers, availability, cost related to geographic locations, and response time
- Platform-as-a-Service (PaaS):
  - Integration, scalability, billing, environment of deployment (licenses, patches, versions, upgrade capability, federation, etc.), servers, browsers, number of developers

# Expected SLA Parameters *(contd…)*

- Software-as-a-Service (SaaS):
  - Reliability, usability, scalability, availability, customizability, Response time
- Storage-as-a-Service :
  - Geographic location, scalability, storage space, storage billing, security, privacy, backup, fault tolerance/resilience, recovery, system throughput, transferring bandwidth, data life cycle management

# Thank You!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

# Cloud Computing : Economics

**Prof. Soumya K Ghosh**
Department of Computer Science and Engineering
IIT KHARAGPUR

# Cloud Properties: Economic Viewpoint

- **C**ommon Infrastructure
  - pooled, standardized resources, with benefits generated by statistical multiplexing.
- **L**ocation-independence
  - ubiquitous availability meeting performance requirements, with benefits deriving from latency reduction and user experience enhancement.
- **O**nline connectivity
  - an enabler of other attributes ensuring service access. Costs and performance impacts of network architectures can be quantified using traditional methods.

# Cloud Properties: Economic Viewpoint Contd…

- **U**tility pricing
  - usage-sensitive or pay-per-use pricing, with benefits applying in environments with variable demand levels.
- on-**D**emand Resource**s**
  - scalable, elastic resources provisioned and de-provisioned without delay or costs associated with change.

# Value of Common Infrastructure

- Economies of scale
  - Reduced overhead costs
  - Buyer power through volume purchasing
- Statistics of Scale
  - For infrastructure built to peak requirements:
    - Multiplexing demand → higher utilization
    - Lower cost per delivered resource than unconsolidated workloads
  - For infrastructure built to less than peak:
    - Multiplexing demand → reduce the unserved demand
    - Lower loss of revenue or a Service-Level agreement violation payout.

# A Useful Measure of "Smoothness"

- The coefficient of variation $C_V$
  - ≠ the variance $\sigma^2$ nor the correlation coefficient
- Ratio of the standard deviation $\sigma$ to the absolute value of the mean $|\mu|$
- "Smoother" curves:
  - large mean for a given standard deviation
  - or smaller standard deviation for a given mean
- Importance of *smoothness*:
  - a facility with fixed assets servicing highly variable demand will achieve lower utilization than a similar one servicing relatively smooth demand.
- **Multiplexing demand from multiple sources may reduce the coefficient of variation $C_V$**

# Coefficient of variation $C_V$

- $X_1, X_2, \ldots, X_n$ independent random variables for demand
  - Identical standard variation $\sigma$ and mean $\mu$
- Aggregated demand
  - Mean → sum of means: n. $\mu$
  - Variance → sum of variances: n. $\sigma^2$
  - Coefficient of variance → $\frac{\sqrt{n}.\sigma}{n.\mu} = \frac{\sigma}{\sqrt{n}.\mu} = \frac{1}{\sqrt{n}} C_v$
- Adding n independent demands reduces the $C_v$ by $\frac{1}{\sqrt{n}}$
  - Penalty of insufficient/excess resources grows smaller
  - Aggregating 100 workloads bring the penalty to 10%

# But What about Workloads?

- Negative correlation demands
  - X and 1-X  Sum is random variable 1
  - Appropriate selection of customer segments
- Perfectly correlated demands
  - Aggregated demand : $n.X$, varianceofsum:$n^2\sigma^2(X)$
  - Mean: $n.\mu$, standard deviation: $n.\sigma(X)$
  - Coefficient of Variance remains constant
- Simultaneous peaks

# Common Infrastructure in Real World

- Correlated demands:
  - Private, mid-size and large-size providers can experience similar statistics of scale
- Independent demands:
  - Midsize providers can achieve similar statistical economies to an infinitely large provider
- Available data on economy of scale for large providers is mixed
  - use the same COTS computers and components
  - Locating near cheap power supplies
  - Early entrant automation tools → 3rd parties take care of it

# Value of Location Independence

- We used to go to the computers, but applications, services and contents now come to us!
  - Through networks: Wired, wireless, satellite, etc.
- But what about latency?
  - Human response latency: 10s to 100s milliseconds
  - Latency is correlated with:
    - **Distance (Strongly)**
    - Routing algorithms of routers and switches (second order effects)
  - Speed of light in fiber: only 124 miles per millisecond
  - If the Google word suggestion took 2 seconds ☹
  - VOIP with latency of 200ms or more ☹

# Value of Location Independence Contd…

- Supporting a global user base requires a dispersed service architecture
  - Coordination, consistency, availability, partition-tolerance
  - **Investment implications**

# Value of Utility Pricing

- As mentioned before, economy of scale might not be very effective
- But cloud services don't need to be cheaper to be economical!
- Consider a car
  - Buy or lease for INR 10,000/- per day
  - Rent a car for INR 45,000/- a day
  - If you need a car for 2 days in a trip, buying would be much more costly than renting
    - **It depends on the demand**

# Utility Pricing in Detail

| | |
|---|---|
| D(t) | demand for resources 0<t<T |
| P | max (D(t)) : Peak Demand |
| A | Avg (D(t)) : Average Demand |
| B | Baseline (owned) unit cost<br>*[$B_T$ : Total Baseline Cost]* |
| C | Cloud unit cost<br>*[$C_T$ : Total Cloud Cost]* |
| U (=C/B) | Utility Premium<br>*[For rental car example, U=4.5]* |

$C_T = \int_0^T U \times B \times D(t)dt = A \times U \times B \times T$

$B_T = P \times B \times T$

- Because the baseline should handle peak demand

When is cloud cheaper than owning?

$C_T < B_T \rightarrow A \times U \times B \times T < P \times B \times T$

$\rightarrow U < \frac{P}{A}$

- When utility premium is less than ratio of peak demand to Average demand

# Utility Pricing in Real World

- In practice demands are often highly spiky
  - News stories, marketing promotions, product launches, Internet flash floods (Slashdot effect), tax season, Christmas shopping, processing a drone footage for a 1 week border skirmish, etc.
- Often a hybrid model is the best
  - You own a car for daily commute, and rent a car when traveling or when you need a van to move
  - Key factor is again the ratio of peak to average demand
  - But we should also consider other costs
    - Network cost (both fixed costs and usage costs)
    - Interoperability overhead
    - Consider Reliability, accessibility

# Value of on-Demand Services

- Simple Problem: When owning your resources, you will pay a penalty whenever your resources do not match the instantaneous demand
    I. Either pay for unused resources, or suffer the penalty of missing service delivery

D(t) – Instantaneous Demand at time t

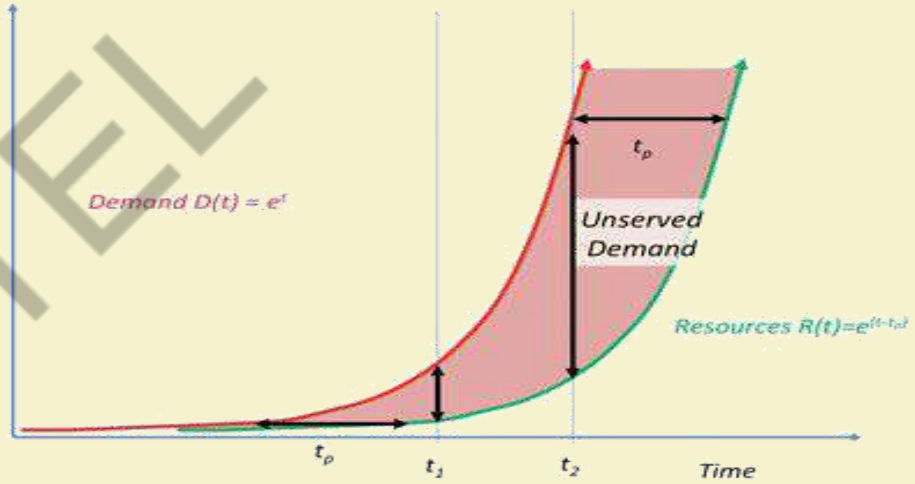
R(t) – Resources at time t

$$\text{Penalty Cost} \alpha \int |D(t) - R(t)| dt$$

- *If demand is flat, penalty = 0*
- *If demand is linear periodic provisioning is acceptable*

# Penalty Costs for Exponential Demand

- Penalty cost $\propto \int |D(t) - R(t)|dt$
- If demand is exponential ($D(t)=e^t$), any fixed provisioning interval ($t_p$) according to the current demands will fall exponentially behind
- $R(t) = e^{t-t_p}$
- $D(t) - R(t) = e^t - e^{t-t_p} = e^t(1 - e^{t_p}) = k_1 e^t$
- Penalty cost $\propto c.k_1 e^t$



Exponential Growth with Continuous Monitoring And Non-Zero Provisioning Interval

# Coefficient of Variation - $C_v$

- A statistical measure of the dispersion of data points in a data series around the mean.
- The coefficient of variation represents the *ratio of the standard deviation to the mean*, and it is a useful statistic for comparing the degree of variation from one data series to another, even if the means are drastically different from each other
- In the investing world, the coefficient of variation allows you to determine how much volatility (risk) you are assuming in comparison to the amount of return you can expect from your investment. In simple language, the lower the ratio of standard deviation to mean return, the better your risk-return tradeoff.

# Assignment 1

Consider the peak computing demand for an organization is 120 units. The demand as a function of time can be expressed as:

$$D(t) = \begin{cases} 50\sin(t), & 0 \le t < \pi/2 \\ 20\sin(t), & \pi/2 \le t < \pi \end{cases}$$

The resource provisioned by the cloud to satisfy current demand at time t is given as:

$$R(t) = D(t) + \delta.\left(\frac{dD(t)}{dt}\right)$$

Where, $\delta$ is the delay in provisioning the extra computing recourse on demand

# Assignment 1 (contd…)

The cost to provision unit cloud resource for unit time is 0.9 units. Calculate the penalty and draw inference.

*[Assume the delay in provisioning is $\pi/_{12}$ time units and minimum demand is 0]*

(Penalty: Either pay for unused resource or missing service delivery)

# Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

# Cloud Computing : *Managing Data*



**Prof. Soumya K Ghosh**
Department of Computer Science and Engineering
IIT KHARAGPUR

# Introduction

- Relational database
  - Default data storage and retrieval mechanism since 80s
  - Efficient in: transaction processing
  - Example: System R, Ingres, etc.
  - Replaced hierarchical and network databases
- For scalable web search service:
  - Google File System (GFS)
    - Massively parallel and fault tolerant distributed file system
  - BigTable
    - Organizes data
    - Similar to column-oriented databases (e.g. Vertica)
  - MapReduce
    - Parallel programming paradigm

# Introduction Contd…

- Suitable for:
  - Large volume massively parallel text processing
  - Enterprise analytics
- Similar to BigTable data model are:
  - Google App Engine's **Datastore**
  - Amazon's **SimpleDB**

# Relational Databases

- Users/application programs interact with an RDBMS through SQL
- RDBM parser:
  - Transforms queries into memory and disk-level operations
  - Optimizes execution time
- Disk-space management layer:
  - Stores data records on pages of contiguous memory blocks
  - Pages are fetched from disk into memory as requested using pre-fetching and page replacement policies

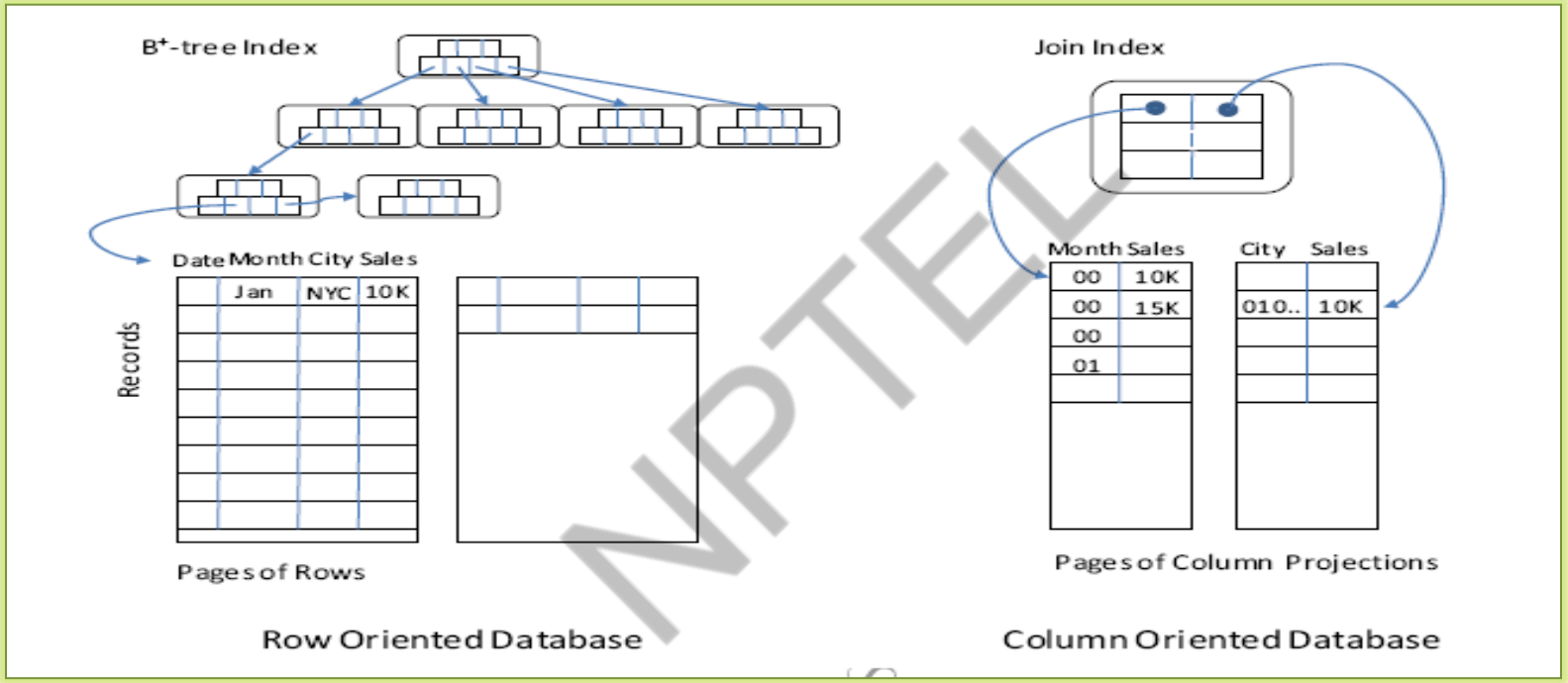
# Relational Databases Contd…

- Database file system layer:
  - Independent of OS file system
  - Reason:
    - To have full control on retaining or releasing a page in memory
    - Files used by the DB may span multiple disks to handle large storage
  - Uses parallel I/O systems, viz. RAID disk arrays or multi-processor clusters

# Data Storage Techniques

- Row-oriented storage
  - Optimal for write-oriented operations viz. transaction processing applications
  - Relational records: stored on contiguous disk pages
  - Accessed through indexes (primary index) on specified columns
  - Example: $B^+$- tree like storage
- Column-oriented storage
  - Efficient for data-warehouse workloads
    - Aggregation of **measure** columns need to be performed based on values from **dimension** columns
    - Projection of a table is stored as sorted by dimension values
    - Require multiple "join indexes"
      - If different projections are to be indexed in sorted order

# Data Storage Techniques Contd…



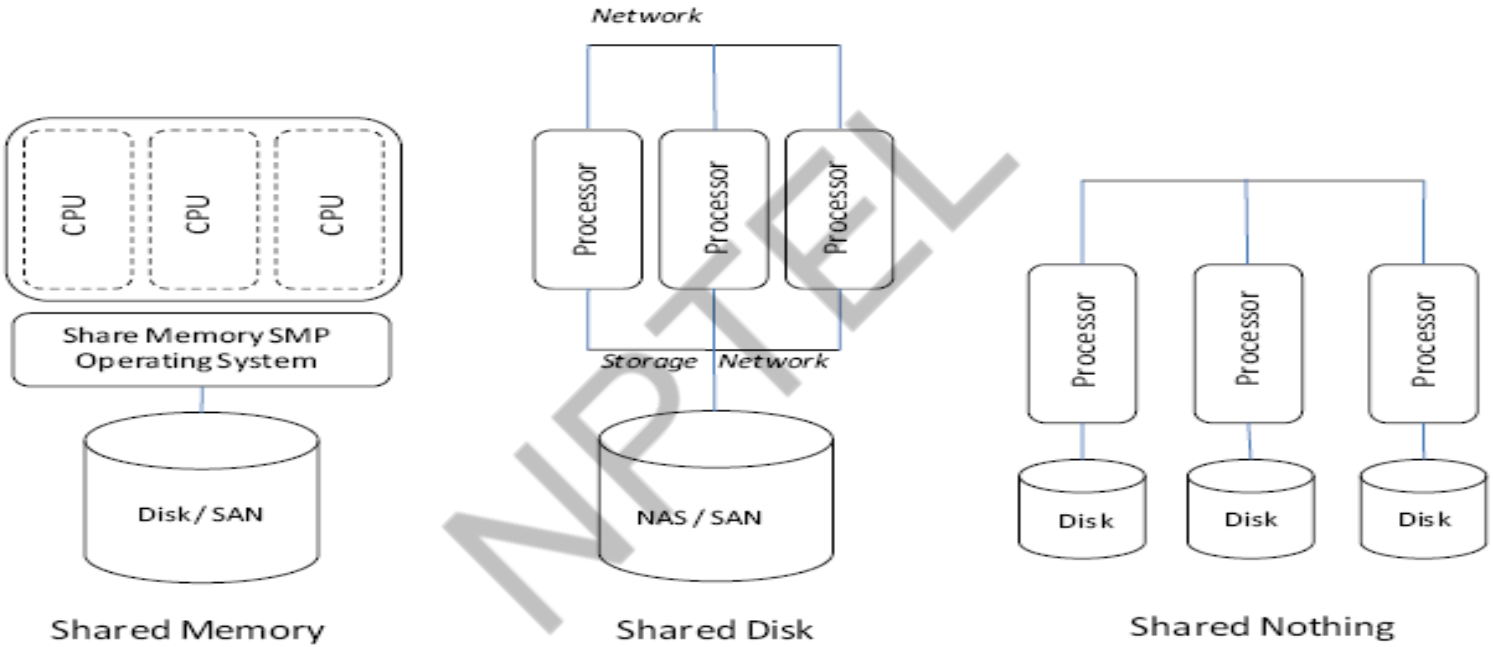



*Source: "Enterprise Cloud Computing" by Gautam Shroff*

# Parallel Database Architectures

- Shared memory
  - Suitable for servers with multiple CPUs
  - Memory address space is shared and managed by a symmetric multi-processing (SMP) operating system
  - SMP:
    - Schedules processes in parallel exploiting all the processors
- Shared nothing
  - Cluster of independent servers each with its own disk space
  - Connected by a network
- Shared disk
  - Hybrid architecture
  - Independent server clusters share storage through high-speed network storage viz. NAS (network attached storage) or SAN (storage area network)
  - Clusters are connected to storage via: standard Ethernet, or faster Fiber Channel or Infiniband connections

# Parallel Database Architectures contd…






*Source: "Enterprise Cloud Computing" by Gautam Shroff*

# Advantages of Parallel DB over Relational DB

- Efficient execution of SQL queries by exploiting multiple processors
- For **shared nothing** architecture:
  - Tables are partitioned and distributed across multiple processing nodes
  - SQL optimizer handles distributed joins
- Distributed **two-phase commit** locking for transaction isolation between processors
- Fault tolerant
  - System failures handled by transferring control to “stand-by” system [for transaction processing]
  - Restoring computations [for data warehousing applications]

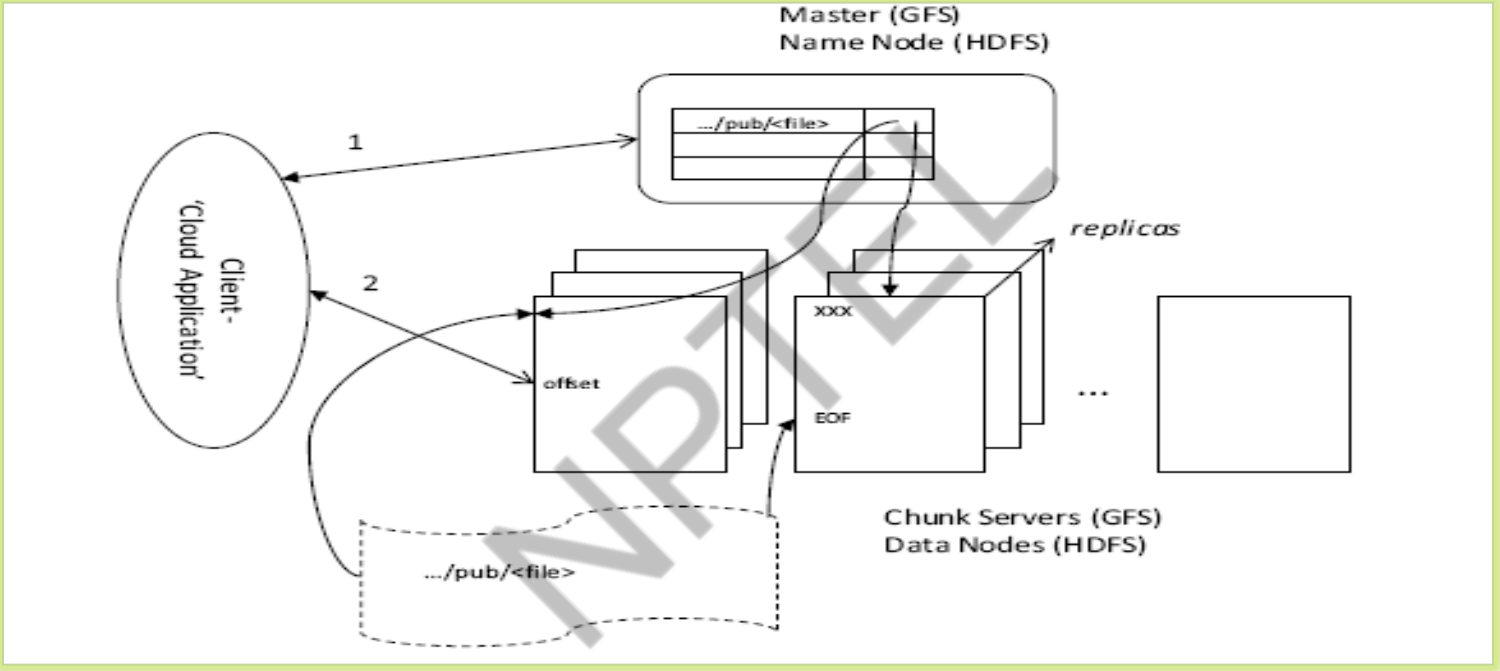
# Advantages of Parallel DB over Relational DB

- Examples of databases capable of handling parallel processing:
  - Traditional transaction processing databases: **Oracle, DB2, SQL Server**
  - Data warehousing databases: **Netezza, Vertica, Teradata**

# Cloud File Systems

- Google File System (GFS)
  - Designed to manage relatively large files using a very large distributed cluster of commodity servers connected by a high-speed network
  - Handles:
    - Failures even during reading or writing of individual files
    - Fault tolerant: a necessity
      - $p$(system failure) = 1-(1-$p$(component failure))$^N$ →1 (for large N)
    - Support parallel reads, writes and appends by multiple simultaneous client programs
- Hadoop Distributed File System (HDFS)
  - Open source implementation of GFS architecture
  - Available on Amazon EC2 cloud platform

# GFS Architecture




*Source: "Enterprise Cloud Computing" by Gautam Shroff*

# GFS Architecture Contd…

- Single Master controls file namespace
- Large files are broken up into **chunks** (GFS) or **blocks** (HDFS)
- Typical size of each chunk: 64 MB
  - Stored on commodity (Linux) servers called **Chunk servers** (GFS) or **Data nodes** (HDFS)
  - Replicated **three** times on different:
    - Physical rack
    - Network segment

# Read Operation in GFS

- Client program sends the full path and offset of a file to the **Master** (GFS) or **Name Node** (HDFS)
- Master replies with meta-data for ***one*** of replicas of the chunk where this data is found.
- Client caches the meta-data for faster access
- It reads data from the designated chunk server

# Write/Append Operation in GFS

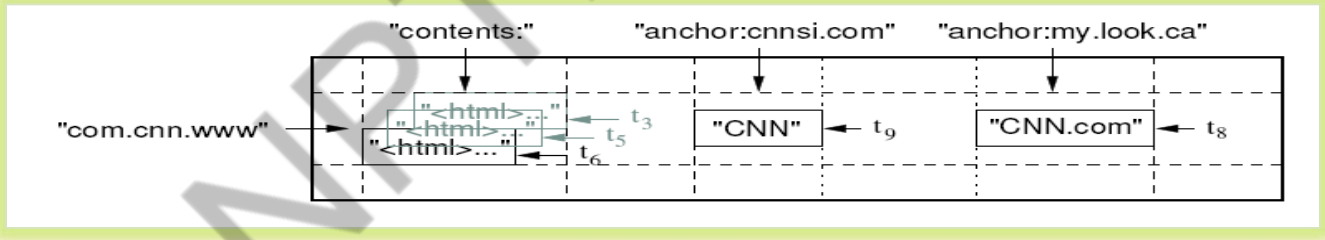
- Client program sends the full path of a file to the **Master** (GFS) or **Name Node** (HDFS)
- Master replies with meta-data for **all** of replicas of the chunk where this data is found.
- Client send data to be appended to all chunk servers
- Chunk server acknowledge the receipt of this data
- Master designates one of these chunk servers as **primary**
- Primary chunk server appends its copy of data into the chunk by choosing an offset
  - Appending can also be done beyond **EOF** to account for multiple simultaneous writers
- Sends the offset to each replica
- If all replicas do not succeed in writing at the designated offset, the primary retries

# Fault Tolerance in GFS

- Master maintains regular communication with chunk servers
  - Heartbeat messages
- In case of failures:
  - Chunk server's meta-data is updated to reflect failure
  - For failure of primary chunk server, the master assigns a new primary
  - Clients occasionally will try to this failed chunk server
    - Update their meta-data from master and retry

# BigTable

- Distributed structured storage system built on GFS
- Sparse, persistent, multi-dimensional sorted map (**key-value pairs**)
- Data is accessed by:
  - Row key
  - Column key
  - Timestamp

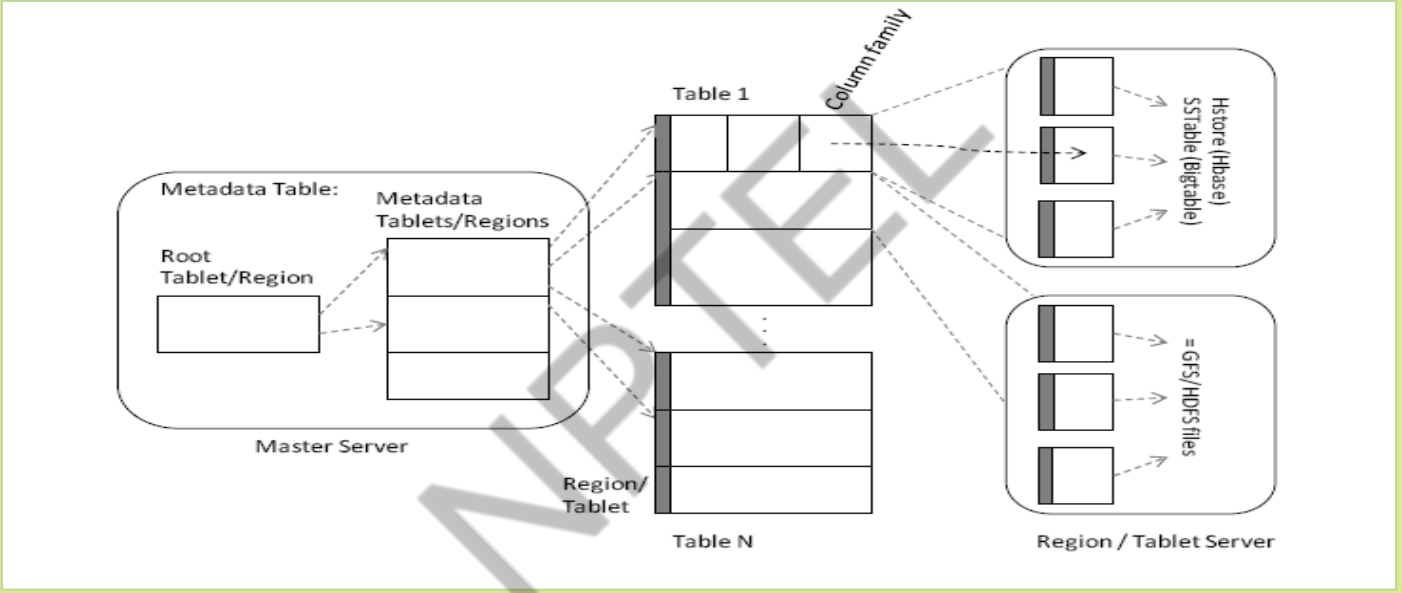



*Source: "Enterprise Cloud Computing" by Gautam Shroff*

# BigTable Contd…

- Each column can store arbitrary **name-value** pairs in the form: ***column-family : label***
- Set of possible column-families for a table is fixed when it is created
- Labels within a column family can be created dynamically and at any time
- Each BigTable cell (row, column) can store multiple versions of the data in decreasing order of timestamp
  - As data in each column is stored together, they can be accessed efficiently

# BigTable Storage




*Source: "Enterprise Cloud Computing" by Gautam Shroff*

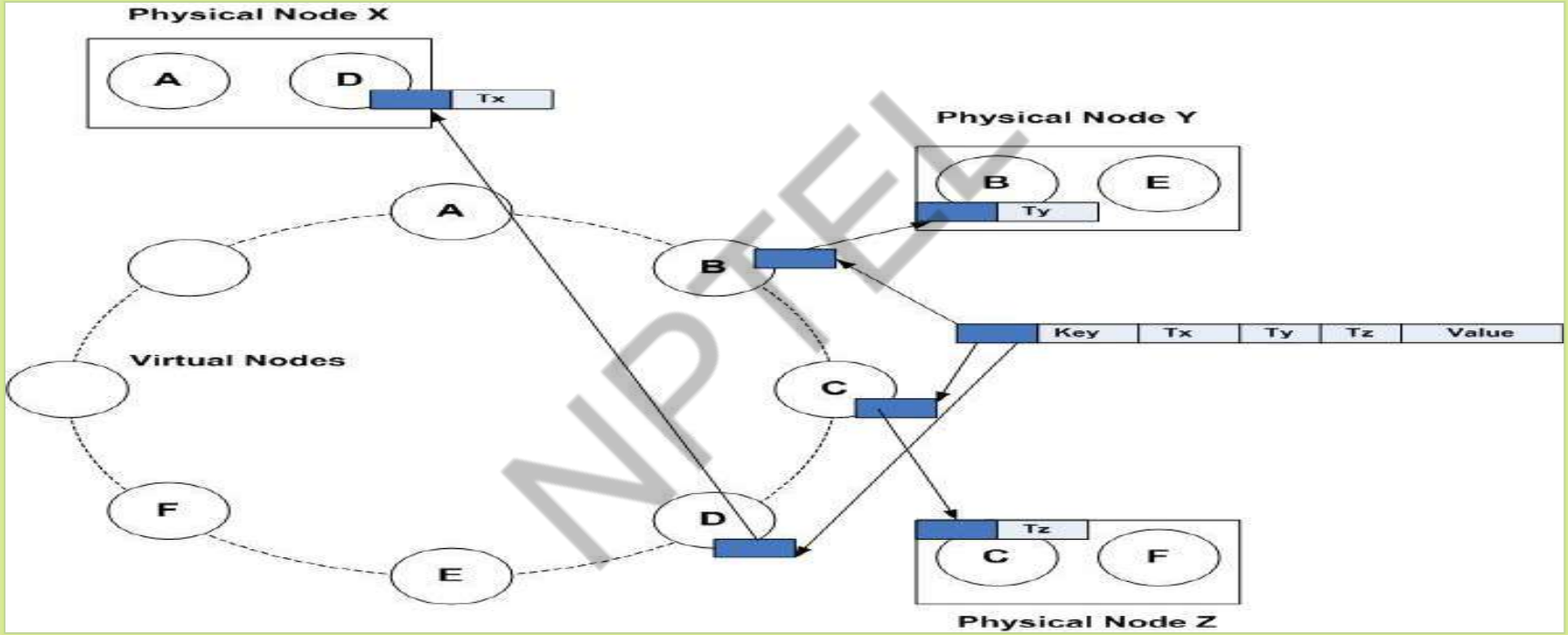
# BigTable Storage Contd…

- Each table is split into different row ranges, called **tablets**
- Each tablet is managed by a **tablet server:**
  - Stores each column family for a given row range in a separate distributed file, called **SSTable**
- A single meta-data table is managed by a **Meta-data server**
  - Locates the tablets of any user table in response to a read/write request
- The meta-data itself can be very large:
  - Meta-data table can be similarly split into multiple tablets
  - A **root tablet** points to other meta-data tablets
- Supports large parallel reads and inserts even simultaneously on the same table
- Insertions done in sorted fashion, and requires more work can simple append

# Dynamo

- Developed by Amazon
- Supports large volume of concurrent updates, each of which could be small in size
  - Different from BigTable: supports bulk reads and writes
- Data model for Dynamo:
  - Simple <key, value> pair
  - Well-suited for Web-based e-commerce applications
  - Not dependent on any underlying distributed file system (for e.g. GFS/HDFS) for:
    - Failure handling
      - Data replication
      - Forwarding write requests to other replicas if the intended one is down
    - Conflict resolution

# Dynamo Architecture



# Dynamo Architecture Contd…

- Objects: <Key, Value> pairs with arbitrary arrays of bytes
- MD5: generates a 128-bit hash value
- Range of this hash function is mapped to a **set of virtual nodes** arranged in a ring
  - Each key gets mapped to one virtual node
- The object is replicated at a **primary** virtual node as well as (N – 1) additional virtual nodes
  - N: number of physical nodes
- Each physical node (server) manages a number of virtual nodes at distributed positions on the ring

# Dynamo Architecture Contd…

- Load balancing for:
  - Transient failures
  - Network partition
- Write request on an object:
  - Executed at one of its virtual nodes
  - Forwards the request to **all** nodes which have the replicas of the object
  - **Quorum protocol**: maintains eventual consistency of the replicas when a large number of concurrent reads & writes take place

# Dynamo Architecture Contd…

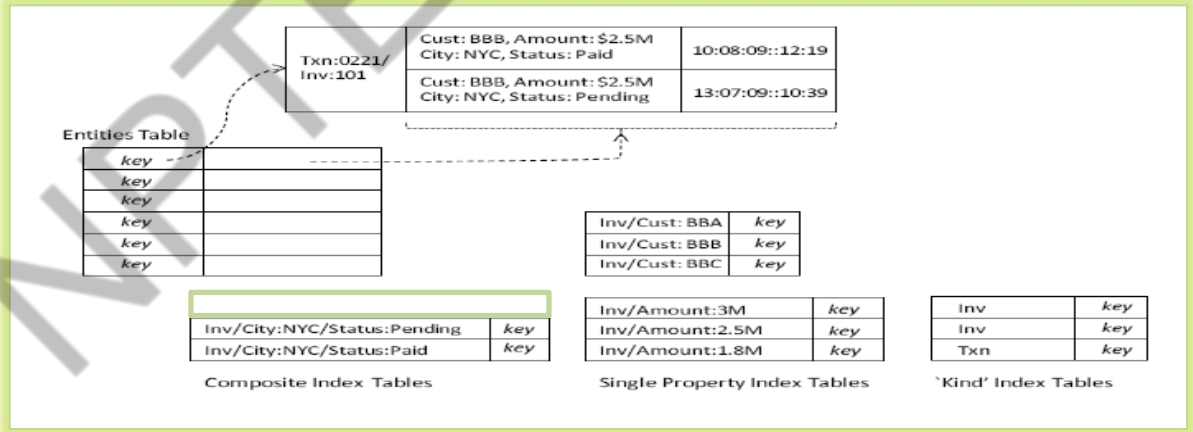
- Distributed object versioning
  - Write creates a new version of an object with its local timestamp incremented
  - Timestamp:
    - Captures history of updates
    - Versions that are superseded by later versions (having larger vector timestamp) are discarded
    - If multiple write operations on same object occurs at the same time, all versions will be maintained and returned to read requests
    - If conflict occurs:
      - Resolution done by application-independent logic

# Dynamo Architecture Contd…

- **Quorum consistent**:
  - Read operation accesses **R** replicas
  - Write operation access **W** replicas
    - If **(R + W) > N** : system is said to be **quorum consistent**
  - Overheads:
    - For efficient write: larger number of replicas to be read
    - For efficient read: larger number of replicas to be written into
- Dynamo:
  - Implemented by different storage engines at node level: Berkley DB (used by Amazon), MySQL, etc.

# Datastore

- Google and Amazon offer simple transactional <Key, Value> pair database stores
  - Google App Engine's Datastore
  - Amazon' SimpleDB
- All entities (objects) in Datastore reside in one BigTable table
  - Does not exploit column-oriented storage
- **Entities table:** store data as one column family




*Source: "Enterprise Cloud Computing" by Gautam Shroff*

# Datastore contd…

- Multiple index tables are used to support efficient queries
- BigTable:
  - Horizontally partitioned (also called **sharded**) across disks
  - Sorted lexicographically by the key values
- Beside lexicographic sorting Datastore enables:
  - Efficient execution of **prefix** and **range** queries on key values
- Entities are ‘grouped’ for transaction purpose
  - Keys are lexicographic by group ancestry
    - Entities in the same group: stored close together on disk
- Index tables: support a variety of queries
  - Uses values of entity attributes as keys

# Datastore Contd…

- Automatically created indexes:
  - Single-Property indexes
    - Supports efficient lookup of the records with **WHERE** clause
  - 'Kind' indexes
    - Supports efficient lookup of queries of form **SELECT ALL**
- Configurable indexes
  - Composite index:
    - Retrieves more complex queries
- Query execution
  - Indexes with highest selectivity is chosen

# Thank You!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

# Cloud Computing :
# *Introduction to MapReduce*

**Prof. Soumya K Ghosh**

**Department of Computer Science and Engineering**

**IIT KHARAGPUR**

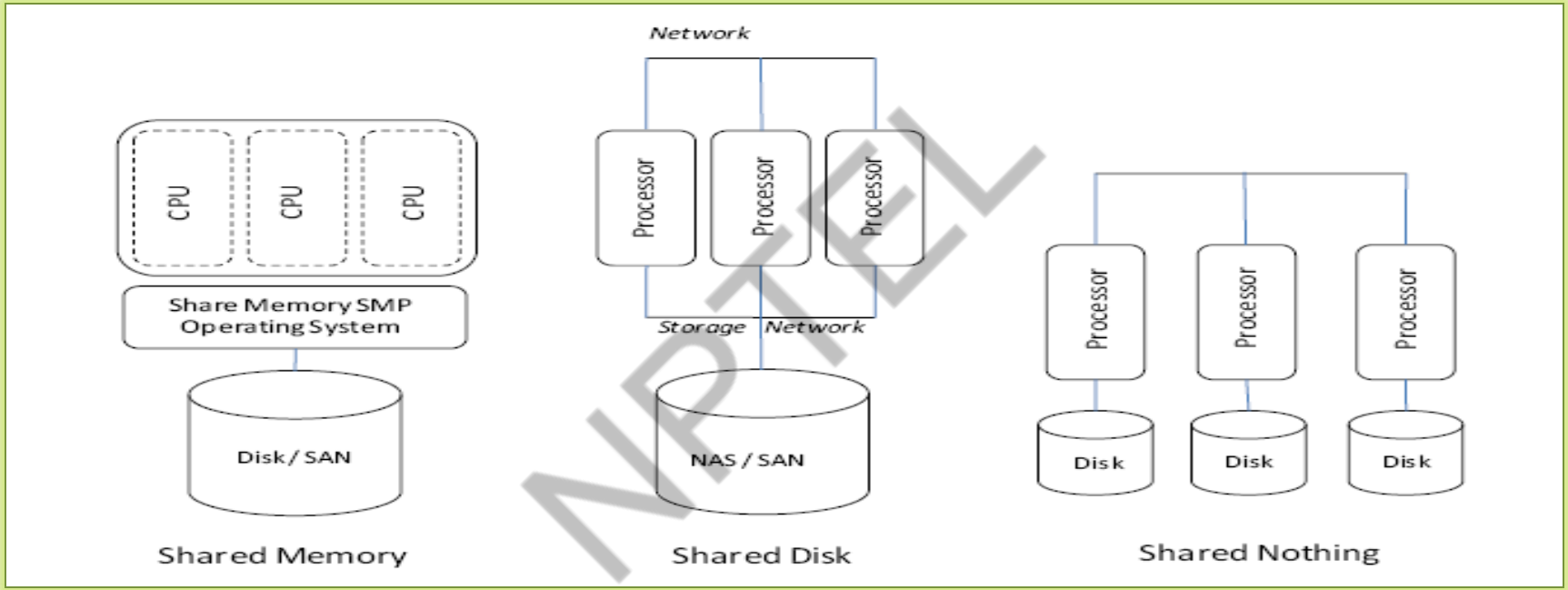
# Introduction

- MapReduce: programming model developed at Google
- Objective:
  - Implement large scale search
  - Text processing on massively scalable web data stored using BigTable and GFS distributed file system
- Designed for processing and generating large volumes of data via massively parallel computations, utilizing tens of thousands of processors at a time
- Fault tolerant: ensure progress of computation even if processors and networks fail
- Example:
  - Hadoop: open source implementation of MapReduce (developed at Yahoo!)
  - Available on pre-packaged AMIs on Amazon EC2 cloud platform

# Parallel Computing

- Different models of parallel computing
  - Nature and evolution of multiprocessor computer architecture
  - Shared-memory model
    - Assumes that any processor can access any memory location
    - Unequal latency
  - Distributed-memory model
    - Each processor can access only its own memory and communicates with other processors using message passing
- Parallel computing:
  - Developed for compute intensive scientific tasks
  - Later found application in the database arena
    - Shared-memory
    - Shared-disk
    - Shared-nothing

# Parallel Database Architectures






*Source: "Enterprise Cloud Computing" by Gautam Shroff*

# Parallel Database Architectures Contd…

- Shared memory
  - Suitable for servers with multiple CPUs
  - Memory address space is shared and managed by a symmetric multi-processing (SMP) operating system
  - SMP:
    - Schedules processes in parallel exploiting all the processors
- Shared nothing
  - Cluster of independent servers each with its own disk space
  - Connected by a network
- Shared disk
  - Hybrid architecture
  - Independent server clusters share storage through high-speed network storage viz. NAS (network attached storage) or SAN (storage area network)
  - Clusters are connected to storage via: standard Ethernet, or faster Fiber Channel or Infiniband connections

# Parallel Efficiency

- If a task takes time *T* in uniprocessor system, it should take *T/p* if executed on *p* processors
- Inefficiencies introduced in distributed computation due to:
  - Need for synchronization among processors
  - Overheads of message communication between processors
  - Imbalance in the distribution of work to processors
- *Parallel efficiency* of an algorithm is defined as:

$$\epsilon = \frac{T}{p\,T_p}.$$

**Scalable** parallel implementation

- parallel efficiency remains constant as the size of data is increased along with a corresponding increase in processors
- parallel efficiency increases with the size of data for a fixed number of processors

# Illustration

- **Problem**: Consider a very large collection of documents, say web pages crawled from the entire Internet. The problem is to determine the frequency (i.e., total number of occurrences) of each word in this collection. Thus, if there are *n* documents and *m* distinct words, we wish to determine *m* frequencies, one for each word.
- Two approaches:
  - Let each processor compute the frequencies for *m/p* words
  - Let each processor compute the frequencies of *m* words across *n/p* documents, followed by all the processors summing their results
- Parallel computing is implemented as a distributed-memory model with a shared disk, so that each processor is able to access any document from disk in parallel with no contention

# Illustration Contd…

- Time to read each word from the document = Time to send the word to another processor via inter-process communication = $c$
- Time to add to a running total of frequencies -> negligible
- Each word occurs $f$ times in a document (on average)
- Time for computing all $m$ frequencies with a single processor = $n \times m \times f \times c$
- First approach:
  - Each processor reads at most $n \times m/p \times f$ times
  - Parallel efficiency is calculated as:
  - Efficiency falls with increasing $p$
  - *Not scalable*

$$\epsilon_a = \frac{nmfc}{pnmfc} = \frac{1}{p}.$$

# Illustration Contd…

- Second approach
  - Number of reads performed by each processor = $n/p \times m \times f$
  - Time taken to read = $n/p \times m \times f \times c$
  - Time taken to write partial frequencies of m-words in parallel to disk = $c \times m$
  - Time taken to communicate partial frequencies to $(p - 1)$ processors and then locally adding $p$ sub-vectors to generate $1/p$ of final m-vector of frequencies = $p \times (m/p) \times c$
  - Parallel efficiency is computed as:

$$\epsilon_b = \frac{nmfc}{p\left(\frac{n}{p}mfc + +cm + p\frac{m}{p}c\right)} = \frac{nf}{nf + 2p} = \frac{1}{1 + \frac{2p}{nf}}.$$

# Illustration Contd…

- Since $p << nf$, efficiency of second approach is higher than that of first
- In fist approach, each processor is reading many words that it need not read, resulting in wasted work
- In the second approach every read is useful in that it results in a computation that contributes to the final answer
- Scalable
  - Efficiency remains constant as both $n$ and $p$ increases proportionally
  - Efficiency tends to 1 for fixed $p$ and gradually increased $n$

# MapReduce Model

- Parallel programming abstraction
- Used by many different parallel applications which carry out large-scale computation involving thousands of processors
- Leverages a common underlying fault-tolerant implementation
- Two phases of MapReduce:
  - Map operation
  - Reduce operation
- A configurable number of M ‘mapper’ processors and R ‘reducer’ processors are assigned to work on the problem
- Computation is coordinated by a single master process

# MapReduce Model Contd…

- Map phase:
  - Each mapper reads approximately *1/M* of the input from the global file system, using locations given by the master
  - Map operation consists of transforming one set of key-value pairs to another:

$$\text{Map:} \quad (k_1, v_1) \rightarrow [(k_2, v_2)].$$

  - Each mapper writes computation results in one file per reducer
  - Files are sorted by a key and stored to the local file system
  - The master keeps track of the location of these files

# MapReduce Model

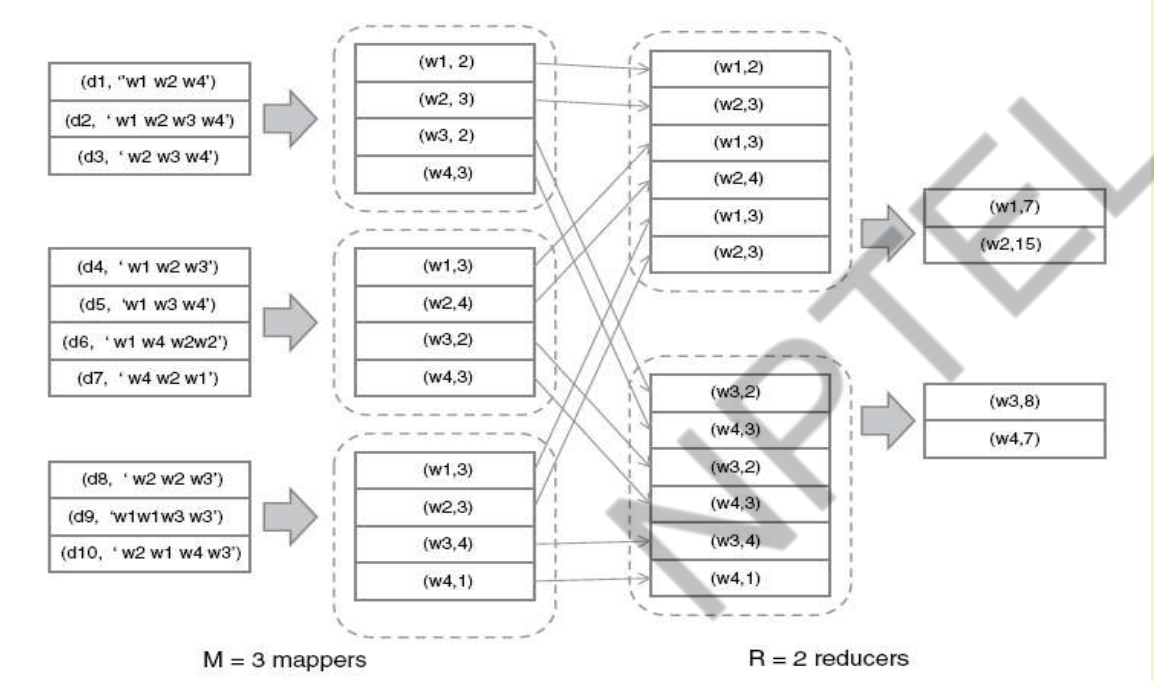
Contd…

- **Reduce phase:**
  - The master informs the reducers where the partial computations have been stored on local files of respective mappers
  - Reducers make remote procedure call requests to the mappers to fetch the files
  - Each reducer groups the results of the map step using the same key and performs a function *f* on the list of values that correspond to these key value:

$$\text{Reduce:} \quad (k_2, [v_2]) \rightarrow (k_2, f([v_2])).$$

  - Final results are written back to the GFS file system

# MapReduce: Example




- 3 mappers; 2 reducers
- Map function:

$$(d_k, [w_1 \ldots w_n]) \rightarrow [(w_i, c_i)].$$

- Reduce function:

$$(w_i, [c_i]) \rightarrow \left(w_i, \sum_i c_i\right)$$

# MapReduce: Fault Tolerance

- Heartbeat communication
  - Updates are exchanged regarding the status of tasks assigned to workers
  - Communication exists, but no progress: master duplicate those tasks and assigns to processors who have already completed
- If a mapper fails, the master reassigns the key-range designated to it to another working node for re-execution
  - Re-execution is required as the partial computations are written into local files, rather than GFS file system
- If a reducer fails, only the remaining tasks are reassigned to another node, since the completed tasks are already written back into GFS

# MapReduce: Efficiency

- General computation task on a volume of data *D*
- Takes *wD* time on a uniprocessor (time to read data from disk + performing computation + time to write back to disk)
- Time to read/write one word from/to disk = *c*
- Now, the computational task is decomposed into map and reduce stages as follows:
  - Map stage:
    - Mapping time = $\mathbf{c_m D}$
    - Data produced as output = $\mathbf{\sigma D}$
  - Reduce stage:
    - Reducing time = $\mathbf{c_r \sigma D}$
    - Data produced as output = $\mathbf{\sigma \mu D}$

# MapReduce: Efficiency Contd…

- Considering no overheads in decomposing a task into a map and a reduce stages, we have the following relation:

$$wD = cD + cmD + cr\sigma D + c\sigma\mu D$$

- Now, we use *P* processors that serve as both mapper and reducers in respective phases to solve the problem
- Additional overhead:
  - Each mapper writes to its local disk followed by each reducer remotely reading from the local disk of each mapper
- For analysis purpose: time to read a word locally or remotely is same
- Time to read data from disk by each mapper = $\frac{wD}{P}$
- Data produced by each mapper = $\frac{\sigma D}{P}$

# MapReduce: Efficiency Contd…

- Time required to write into local disk = $\frac{c\sigma D}{P}$
- Data read by each reducer from its partition in each of *P* mappers = $\frac{\sigma D}{P^2}$
- The entire exchange can be executed in *P* steps, with each reducer *r* reading from mapper *r + i mod r* in *step i*
- Transfer time from mapper local disk to GFS for each reducer = $\frac{c\sigma D}{P2} \times P = \frac{c\sigma D}{P}$
- Total overhead in parallel implementation due to intermediate disk reads and writes = $\left(\frac{wD}{P} + 2c\,\frac{\sigma D}{P}\right)$
- Parallel efficiency of the MapReduce implementation:

$$\varepsilon_{MR} = \frac{wD}{P\left(\frac{wD}{P} + 2c\,\frac{\sigma D}{P}\right)} = \frac{1}{1 + \frac{2c}{w}\sigma}$$

# MapReduce: Applications

- Indexing a large collection of documents
  - Important aspect in web search as well as handling structured data
  - The map task consists of emitting a word-document/record-id pair for each word: $(\boldsymbol{d_k}, [\boldsymbol{w_1} \ldots \boldsymbol{wn}]) \rightarrow [(\boldsymbol{w_i}, \boldsymbol{dk})]$
  - The reduce step groups the pairs by word and creates an index entry for each word: $[(\boldsymbol{w_i}, \boldsymbol{dk})] \rightarrow (\boldsymbol{wi}, [\boldsymbol{d_{i_1}} \ \ldots \ \boldsymbol{dim}])$

- Relational operations using MapReduce
  - Execute SQL statements (relational joins/group by) on large data sets
  - Advantages over parallel database
    - Large scale
    - Fault-tolerance

# Thank You!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

# CLOUD COMPUTING

## OPENSTACK:


PROF. SOUMYA K. GHOSH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IIT KHARAGPUR

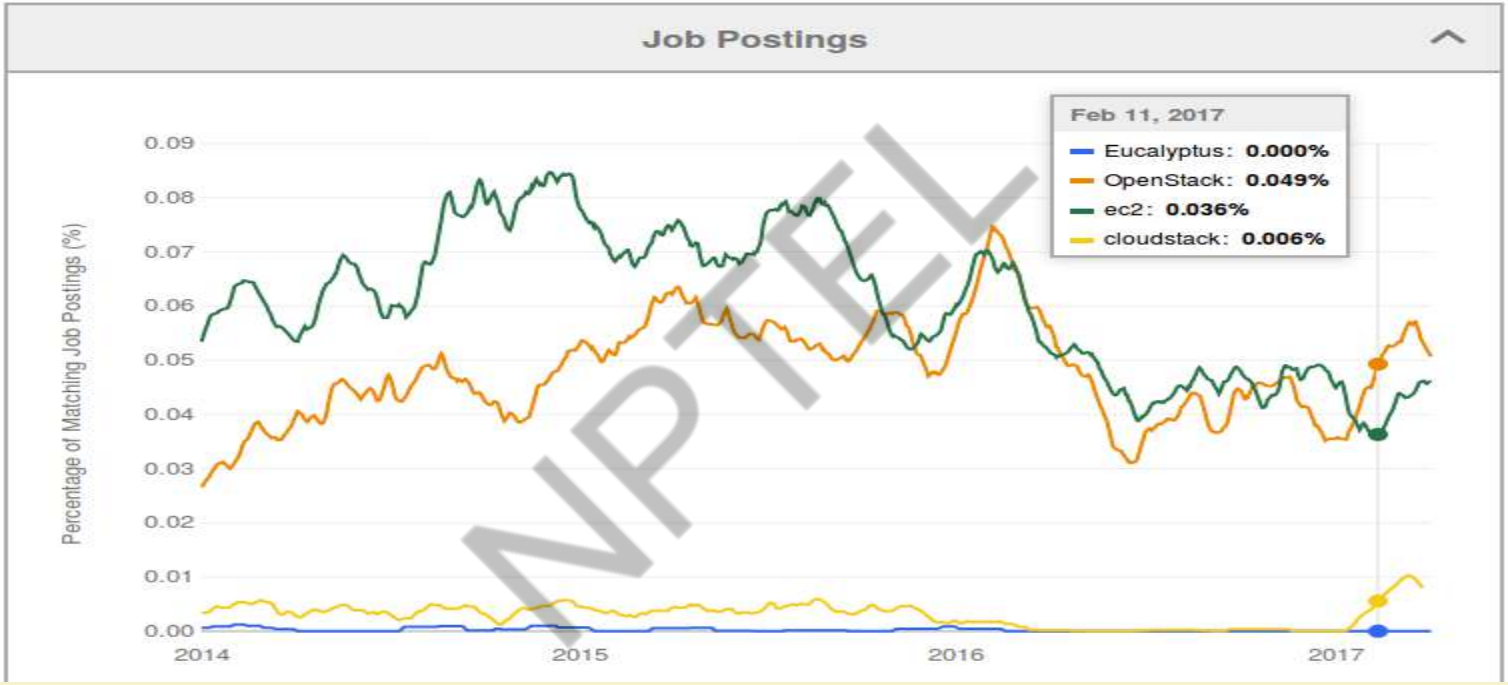
# What is OpenStack?

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.



*Source: OpenStack, http://www.doc.openstack.org*

# Job Trend for Openstack

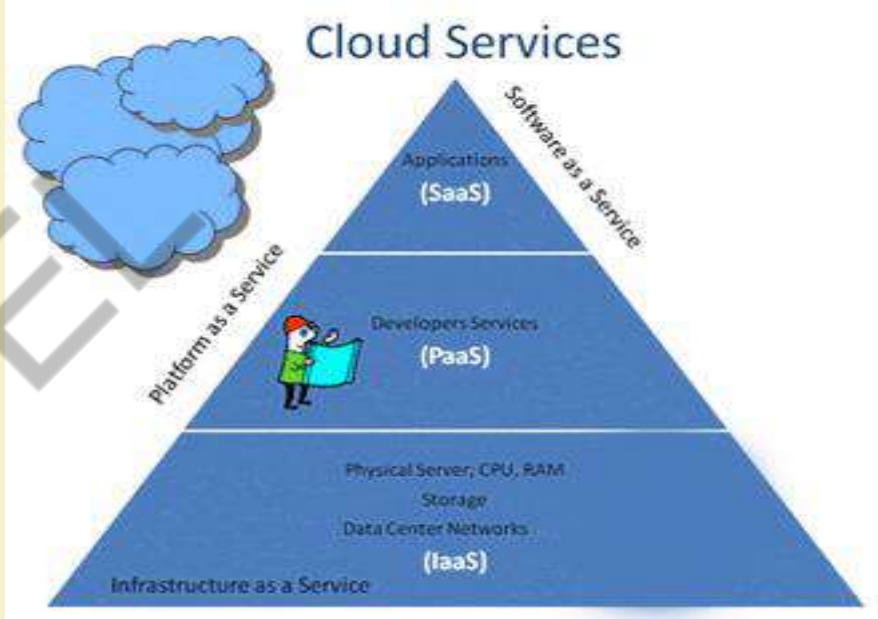





*Source: http://www.indeed.com, Accessed on:July-2017*

# OpenStack Capability

- Software as Service (SaaS)
  - Browser or Thin Client access
- Platform as Service (PaaS)
  - On top of IaaS e.g. Cloud Foundry
- Infrastructure as Service (IaaS)
  - Provision Compute, Network, Storage



# OpenStack Capability

- Virtual Machine (VMs) on demand
  - Provisioning
  - Snapshotting
- Network
- Storage for VMs and arbitrary files
- Multi-tenancy
  - Quotas for different project, users
  - User can be associated with multiple projects

# OpenStack History

| Series | Status | Initial Release Date | Next Phase | EOL Date |
|---|---|---|---|---|
| Queens | *Future* | TBD | | TBD |
| Pike | Under Development | TBD | | TBD |
| Ocata | Phase I – Latest release | 2017-02-22 | Phase II – Maintained release on 2017-08-28 | 2018-02-26 |
| Newton | Phase II – Maintained release | 2016-10-06 | Phase III – Legacy release on 2017-10-09 | 2017-10-11 |
| Mitaka | EOL | 2016-04-07 | | 2017-04-10 |
| Liberty | EOL | 2015-10-15 | | 2016-11-17 |
| Kilo | EOL | 2015-04-30 | | 2016-05-02 |
| Juno | EOL | 2014-10-16 | | 2015-12-07 |
| Icehouse | EOL | 2014-04-17 | | 2015-07-02 |
| Havana | EOL | 2013-10-17 | | 2014-09-30 |
| Grizzly | EOL | 2013-04-04 | | 2014-03-29 |
| Folsom | EOL | 2012-09-27 | | 2013-11-19 |
| Essex | EOL | 2012-04-05 | | 2013-05-06 |
| Diablo | EOL | 2011-09-22 | | 2013-05-06 |
| Cactus | Deprecated | 2011-04-15 | | |
| Bexar | Deprecated | 2011-02-03 | | |
| Austin | Deprecated | 2010-10-21 | | |

**Started as a collaboration between NASA and Rackspace*

# OpenStack Major Components

- Service - Compute
- Project - Nova

Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.

# OpenStack Major Components

- Service - Networking
- Project - Neutron

- Enables *Network-Connectivity-as-a-Service* for other OpenStack services, such as OpenStack Compute.
- Provides an API for users to define networks and the attachments into them.
- Has a pluggable architecture that supports many popular networking vendors and technologies.

# OpenStack Major Components

- Service - Object storage
- Project - Swift

- Stores and retrieves arbitrary unstructured data objects via a RESTFul, HTTP based API.
- It is highly fault tolerant with its data replication and scale-out architecture. Its implementation is not like a file server with mountable directories.
- In this case, it writes objects and files to multiple drives, ensuring the data is replicated across a server cluster.

# OpenStack Major Components

- Service- Block storage
- Project- Cinder

- Provides persistent block storage to running instances.
- Its pluggable driver architecture facilitates the creation and management of block storage devices.

# OpenStack Major Components

- Service - Identity
- Project - Keystone

- Provides an authentication and authorization service for other OpenStack services.
- Provides a catalog of endpoints for all OpenStack services.

# OpenStack Major Components

- Service - Image service
- Project - Glance

- Stores and retrieves virtual machine disk images.
- OpenStack Compute makes use of this during instance provisioning.

# OpenStack Major Components

- Service - Telemetry
- Project - Ceilometer

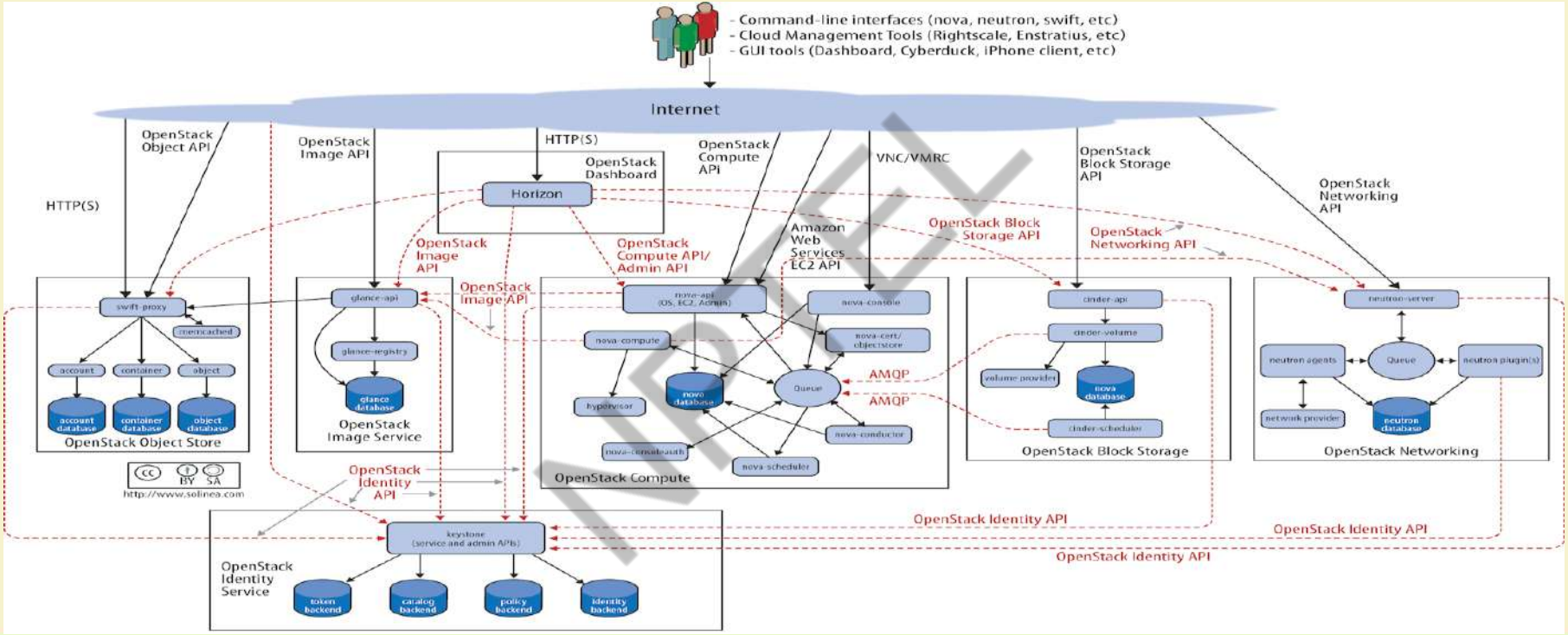
- Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.

# OpenStack Major Components

- Service - Dashboard
- Project - Horizon

- Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.

# Architecture of Openstack



# Openstack Work Flow

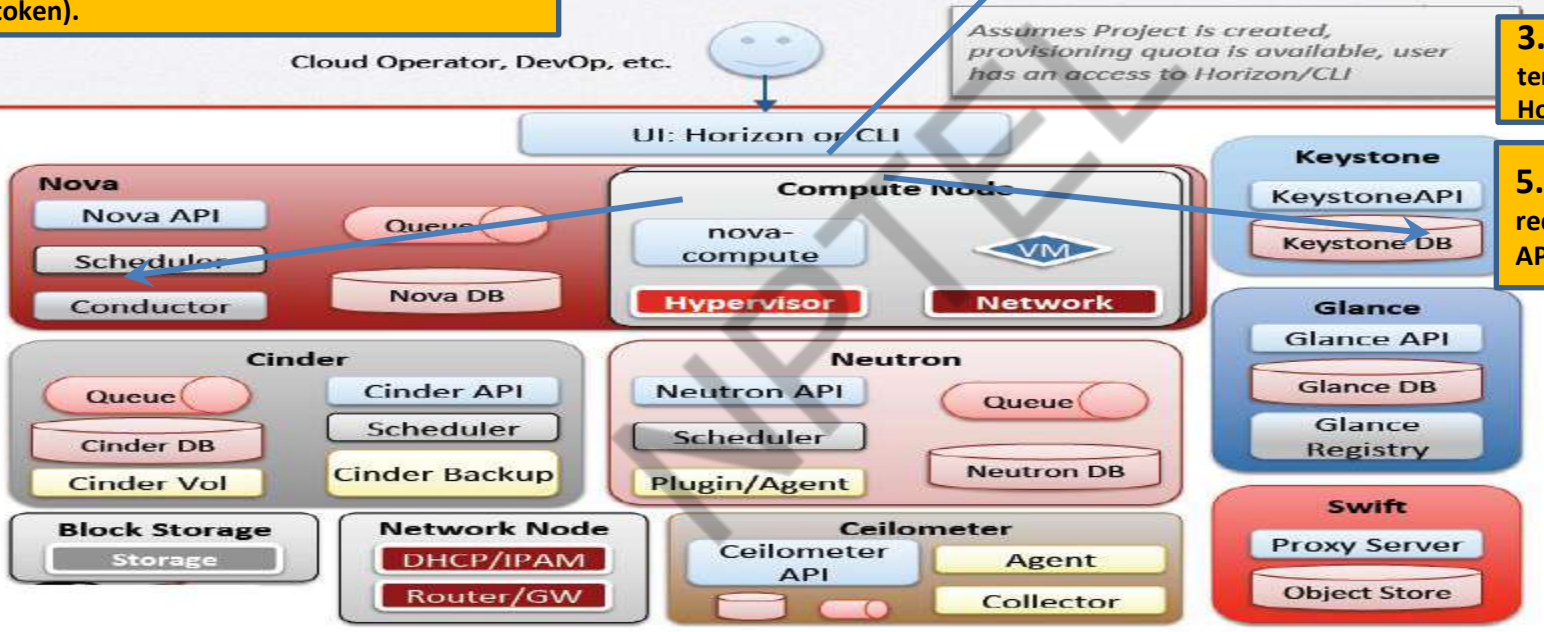
**1.** User logs in to UI Specifies VM params: name,flavor,keys,etc. and hits "Create" button

**2.** Horizon sends HTTP request to Keystone. Auth info is specified in HTTP headers.

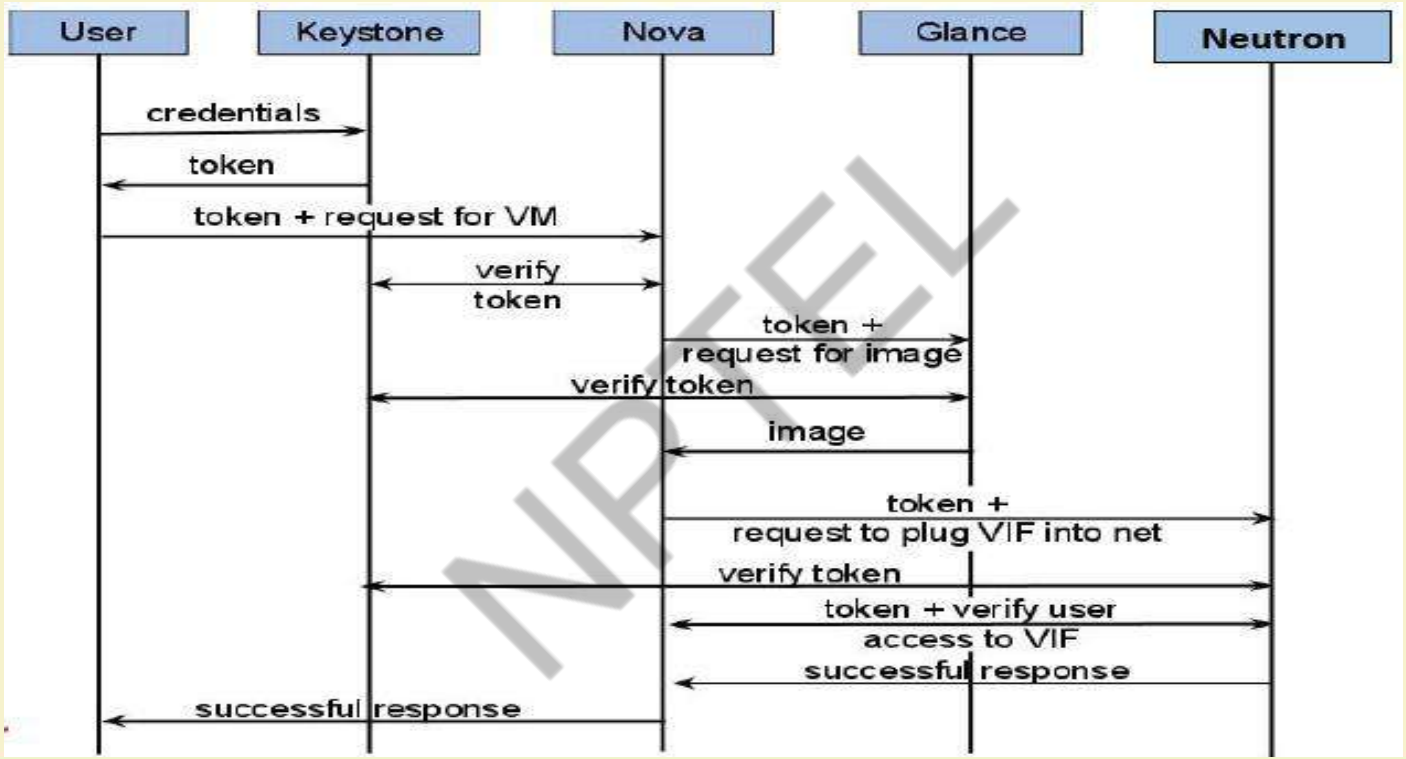
**3.** Keystone sends temporary token back to Horizon via HTTP.

**4.** Keystone sends temporary token back to Horizon via HTTP. Horizon sends POST request to Nova API(signed with given token).

**5.** Nova API sends HTTP request to validate API token to Keystone.



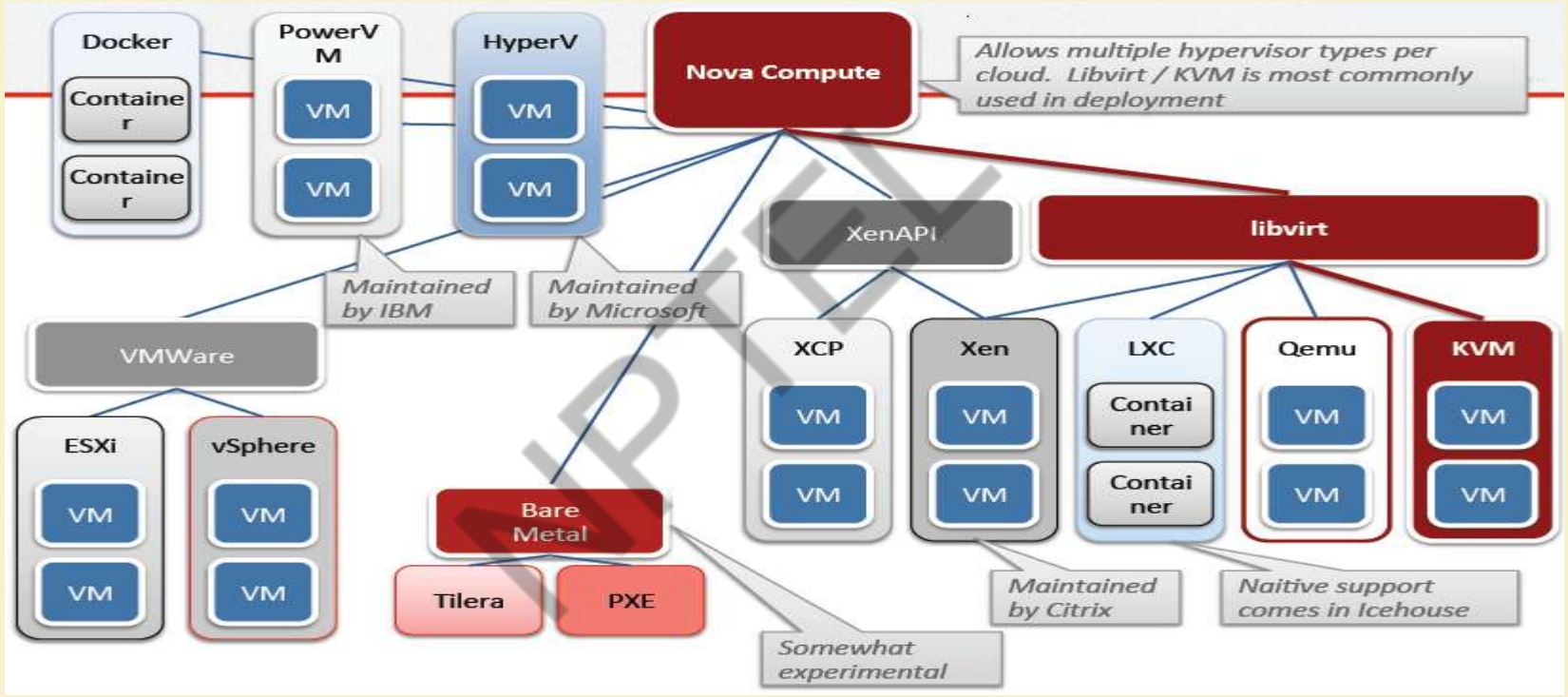
# Auth Token Usage



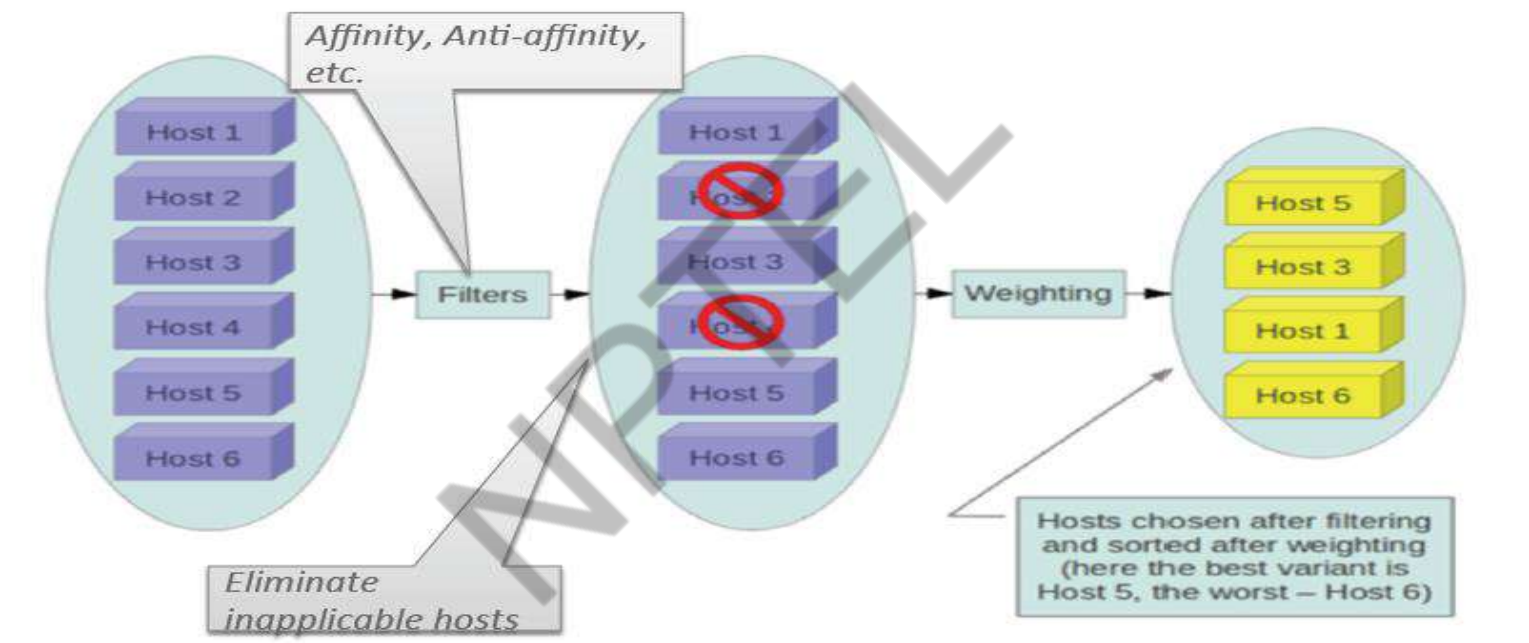
# Provisioning Flow

- Nova API makes rpc.cast to Scheduler. It publishes a short message to scheduler queue with VM info.
- Scheduler picks up the message from MQ.
- Scheduler fetches information about the whole cluster from database, filters, selects compute node and updates DB with its ID
- Scheduler publishes message to the compute queue (based on host ID) to trigger VM provisioning
- Nova Compute gets message from MQ
- Nova Compute makes rpc.call to Nova Conductor for information on VM from DB
- Nova Compute makes a call to Neutron API to provision network for the instance
- Neutron configures IP, gateway, DNS name, L2 connectivity etc.
- It is assumed a volume is already created. Nova Compute contacts Cinder to get volume data. Can also attach volumes after VM is built.

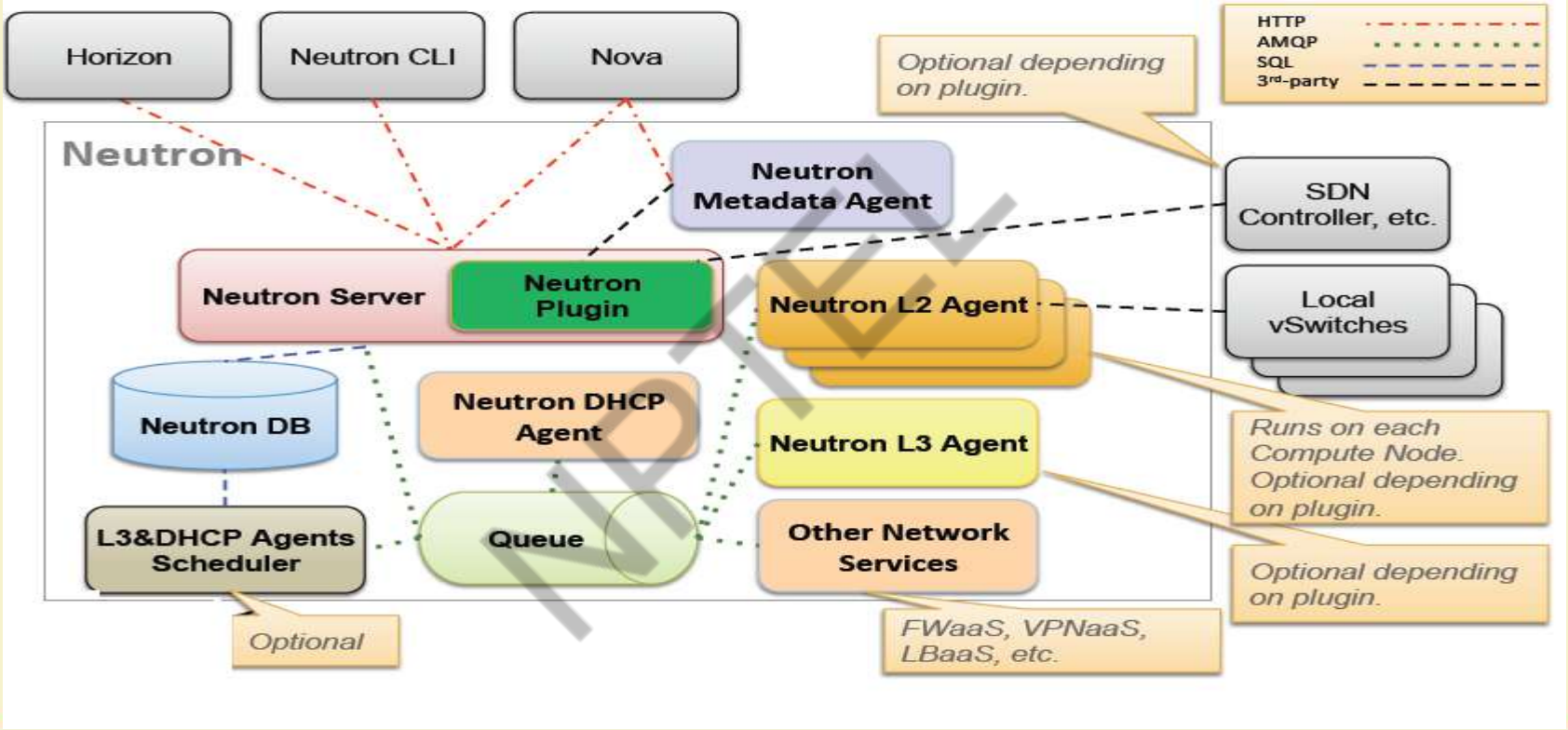
# Nova Compute Driver



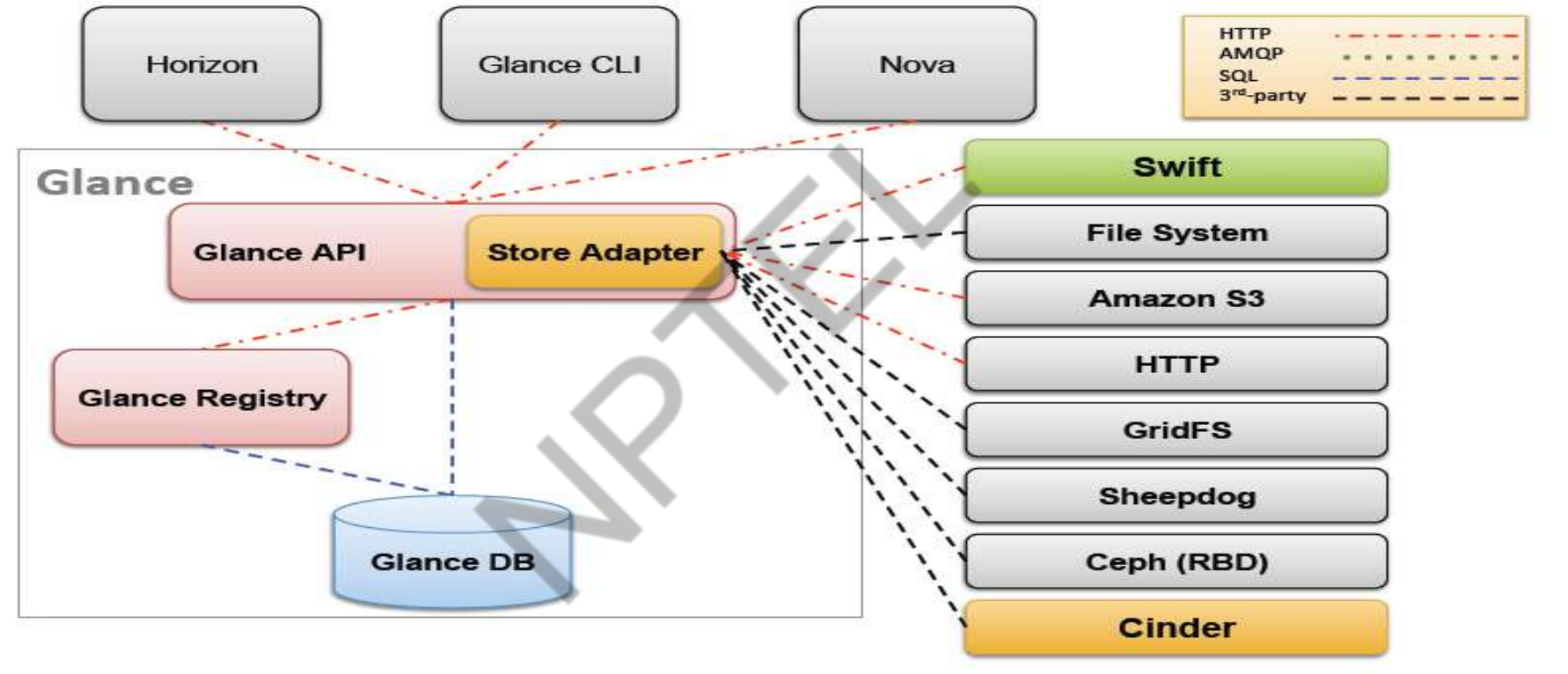
# Nova scheduler filtering



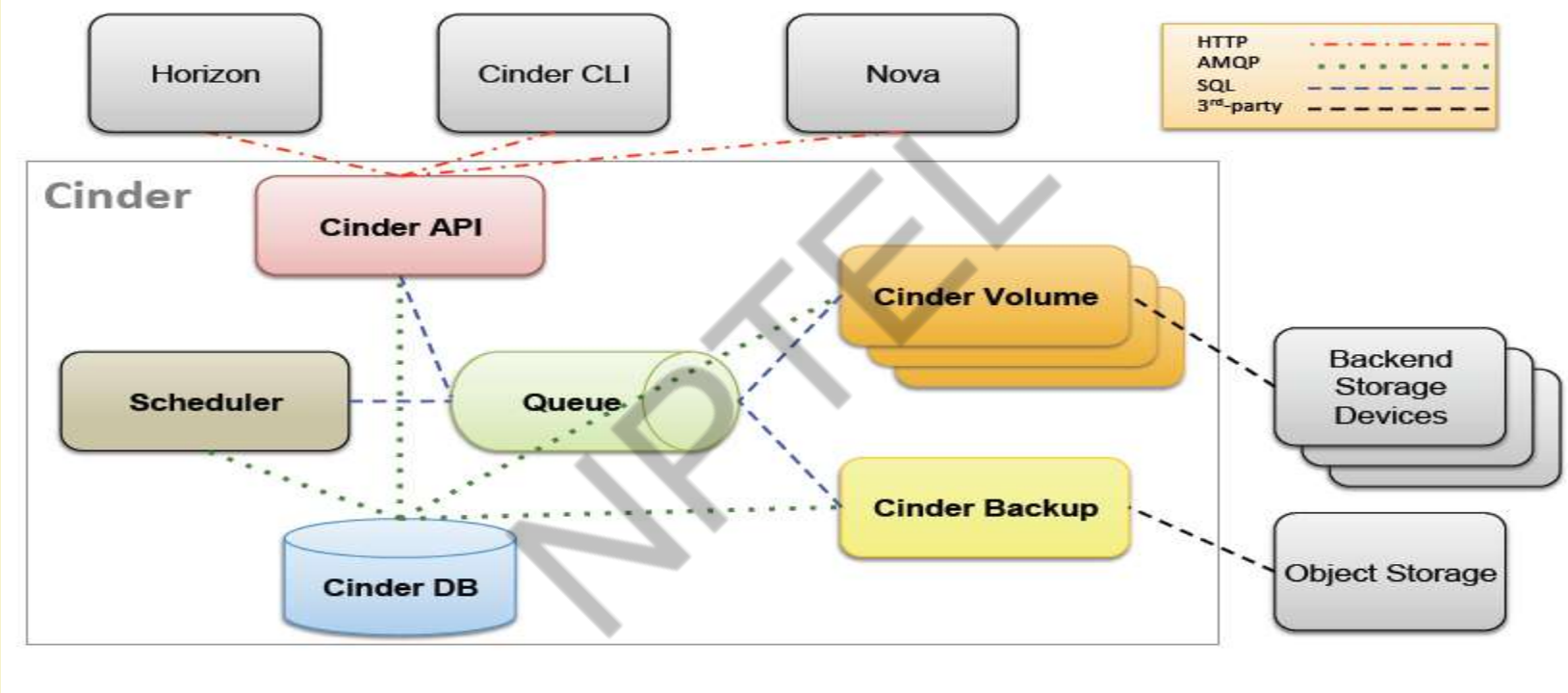
# Neutron Architecture



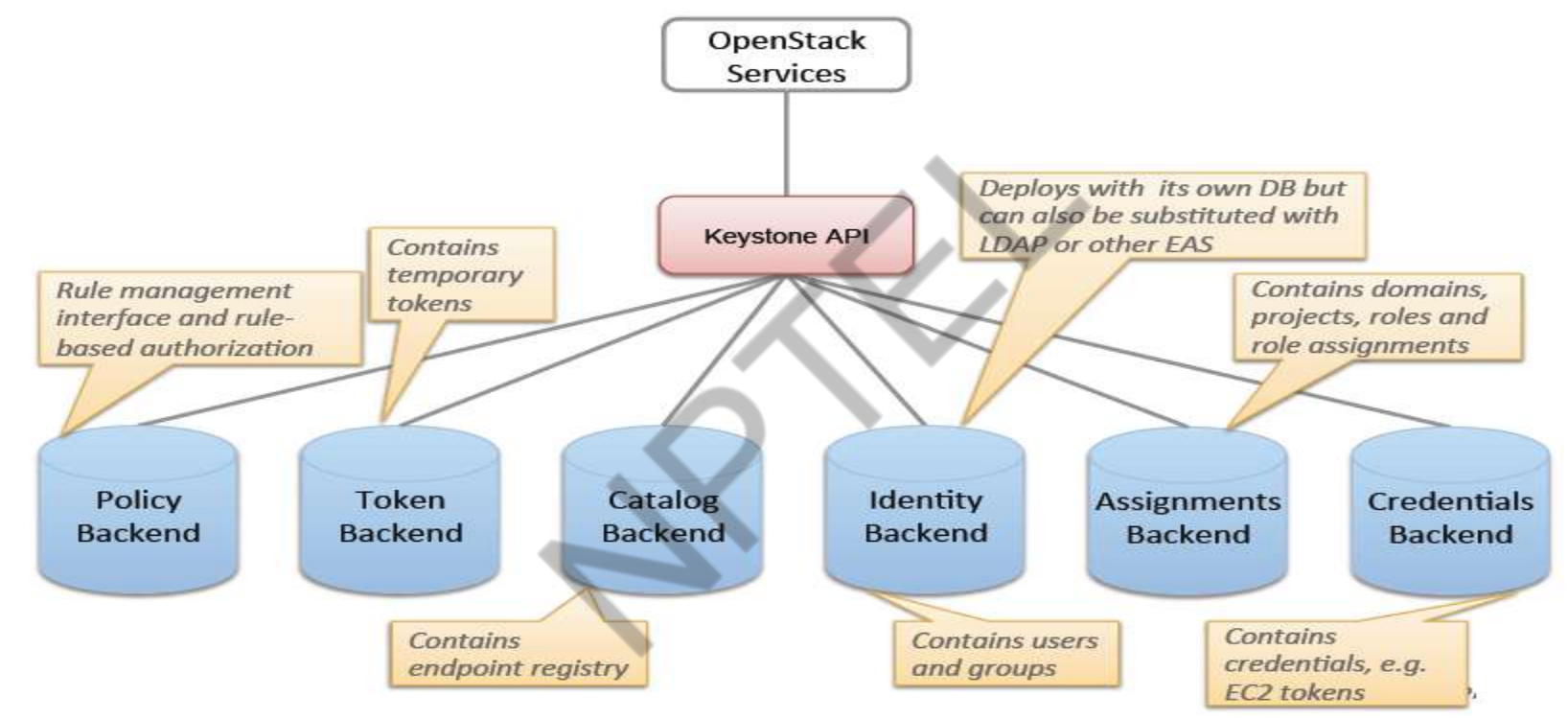
# Glance Architecture



# Cinder Architecture



# Keystone Architecture



# OpenStack Storage Concepts

- **Ephemeral storage:**
  - Persists until VM is terminated
  - Accessible from within VM as local file system
  - Used to run operating system and/or scratch space
  - Managed by Nova
- **Block storage**:
  - Persists until specifically deleted by user
  - Accessible from within VM as a block device (e.g. /dev/vdc)
  - Used to add additional persistent storage to VM and/or run operating system
  - Managed by Cinder
- **Object storage:**
  - Persists until specifically deleted by user
  - Accessible from anywhere
  - Used to add store files, including VM images
  - Managed by Swift

# Summary

- Users log into Horizon and initiates VM creation
- Keystone authorizes
- Nova initiates provisioning and saves state to DB
- Nova Scheduler finds appropriate host
- Neutron configures networking
- Cinder provides block device
- Image URI is looked up through Glance
- Image is retrieved via Swift
- VM is rendered by Hypervisor

# Thank You!