CS 4296 Could Computing

Table of contents

1. Fundamentals . . . 7
   1.1. Outline . . . 7
   1.2. Cloud Definition . . . 7
   1.3. Key Cloud Concepts and Characteristics . . . 7
      1.3.1. On-demand self-service . . . 7
      1.3.2. Broad network access . . . 7
      1.3.3. Resource pooling . . . 7
      1.3.4. Rapid elasticity . . . 8
      1.3.5. Measured service . . . 8
   1.4. Why use cloud? . . . 8
   1.5. Cloud deployment models . . . 8
      1.5.1. Public cloud . . . 8
      1.5.2. Private cloud . . . 8
      1.5.3. Hybrid cloud . . . 9
   1.6. Issues of cloud . . . 9
2. Service models . . . 10
   2.1. Outline . . . 10
   2.2. IaaS . . . 10
   2.3. PaaS . . . 10
   2.4. SaaS . . . 10
3. Virtualization . . . 11
   3.1. Outline . . . 11
   3.2. Concepts . . . 11
      3.2.1. Some terms . . . 11
   3.3. Virtualization architecture . . . 11
      3.3.1. Hosted architecture . . . 11
      3.3.2. Hypervisor architecture . . . 12
   3.4. OS review . . . 12
      3.4.1. What is OS? . . . 12
      3.4.2. Four fundamental OS concepts . . . 13
      3.4.3. Protection rings . . . 13
   3.5. CPU virtualization . . . 13
      3.5.1. Full virtualization . . . 14
      3.5.2. Para-virtualization . . . 15
      3.5.3. Hardware-assisted . . . 15
      3.5.4. Summary . . . 17
   3.6. Cloud Infrastructures . . . 17
4. Cluster Scheduling . . . 18
   4.1. Background . . . 18

4.1.1. Trends Observerd . . . 18
4.1.2. Workload: batch/service split . . . 18
4.2. Existing Approaches . . . 18
4.3. Mesos . . . 19
4.3.1. Mesos Goals . . . 19
4.3.2. Design Elements . . . 20
4.4. Omega: shared-state . . . 20
4.5. Job Scheduling Fairness . . . 20
4.5.1. Properties of max-min fairness . . . 20
4.5.2. Desirable properties of max-min fairness . . . 20
4.5.3. Fair share multiple resources for heterogenous demands . . . 21
5. CPU Scheduling . . . 22
5.1. Outline . . . 22
5.2. Motivation . . . 22
5.3. Fairness . . . 22
5.4. Utilization . . . 22
5.4.1. Work-conserving . . . 22
5.4.2. Non work-conserving . . . 22
5.5. Three CPU Schedulers in Xen . . . 23
5.5.1. Borrowed Virtual Time (BVT) . . . 23
5.5.2. Simple earliest deadline first (SEDF) . . . 24
5.5.3. Credit Scheduler . . . 26
6. Data Center Network - Basics, Topology, Traffic . . . 29
6.1. Outline . . . 29
6.2. History . . . 29
6.3. Some basic concepts . . . 29
6.3.1. Building blocks . . . 29
6.3.2. Switching approaches . . . 30
6.3.3. Addressing, routing . . . 30
6.3.4. Multiplexing . . . 30
6.3.5. Statistical multiplexing . . . 30
6.3.6. Performance metrics . . . 30
6.4. Layering architecture . . . 31
6.4.1. Why layering? . . . 31
6.4.2. OSI reference model . . . 31
6.4.3. More popular TCP/IP model . . . 31
6.5. TCP/IP . . . 32
6.5.1. Internet protocol - IP . . . 32
6.5.2. Routing . . . 32
6.5.3. Transport layer . . . 32
6.5.4. TCP overview . . . 32
6.5.5. Reliable transfer . . . 33
6.5.6. TCP flow control . . . 33
6.5.7. TCP congestion control . . . 33
6.6. Topology . . . 34

6.6.1. 2004: four-post cluster . . . 34
6.6.2. 2005: Firrehose 1.0 . . . 35
6.6.3. 2012: Jupiter . . . 35
6.6.4. Facebook's fabric . . . 35
6.6.5. Academia: Fat-tree . . . 36
6.7. Traffic characteristics . . . 36
6.7.1. Flow size . . . 36
6.7.2. Localility . . . 37
7. Data Center Networking - Latency . . . 38
7.1. Outline . . . 38
7.2. Multipath routing . . . 38
7.2.1. Multipath in Fat-tree . . . 38
7.2.2. ECMP . . . 38
7.3. Latency problem . . . 38
7.3.1. Traffic characteristics . . . 39
7.4. Solutions . . . 39
7.4.1. Overview . . . 39
7.4.2. Reduce queue length: DCTCP . . . 39
7.4.3. Prioritize mice flows: pFabric . . . 41
7.4.4. Better multipath routing: RepFlow . . . 41
7.4.5. Congestion-aware LB . . . 41
7.4.6. A strawman design . . . 41
8. Data Center Networking - QoS . . . 43
8.1. Outline . . . 43
8.2. Queueing disciplines . . . 43
8.2.1. FIFO Queueing . . . 43
8.2.2. Round-Robin Queueing . . . 44
8.2.3. Fair Queueing . . . 44
8.2.4. Weighted Fair Queueing (WFQ) . . . 45
8.3. Traffic shaping . . . 45
8.3.1. Motivation . . . 46
8.3.2. Token bucket . . . 46
8.4. Rate delay guarantees . . . 46
8.4.1. Motivation . . . 46
8.4.2. Admission control . . . 47
8.4.3. Single router rate guarantee . . . 47
8.4.4. Network rate guarantee . . . 47
8.4.5. Router delay guarantee . . . 48
8.4.6. Network delay guarantee . . . 48
8.4.7. Rate delay guarantees . . . 48
9. Storage Systems . . . 49
9.1. Outline . . . 49
9.2. NoSQL . . . 49
9.3. CAP theorem . . . 49
9.3.1. Distributed data store . . . 49

9.4. Chord . . . 50
9.4.1. Solution 1 . . . 50
9.4.2. Solution2 - finger table . . . 51
9.5. Amazon Dynamo . . . 52
9.5.1. Node Failures . . . 52
9.5.2. Consistency . . . 52
10. Security . . . 54
10.1. Outline . . . 54
10.2. Principles of security . . . 54
10.3. Plain Text and Cipher Text . . . 54
10.3.1. Tranditional ways . . . 54
10.3.2. Some concepts . . . 54
10.4. Cryptography . . . 55
10.4.1. Symmetric key cryptography . . . 55
10.4.2. Asymmetric key cryptography . . . 55

1. Fundamentals

1.1. Outline

- Real-world examples
- Definitions of Cloud Computing
- Key cloud concepts and characteristics
- Deployment scenarios

1.2. Cloud Definition

Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.[1]

1.3. Key Cloud Concepts and Characteristics

1.3.1. On-demand self-service

A consumer can unilaterally provision computing capabilities, such as servers and network storage, as needed automatically without requiring human interaction with each service provider.

Advantages for consumers: flexible, minimal overhead, quick and easy

1.3.2. Broad network access

Advantages for consumers: "Always-on" experience, like utilities (electricity)

1.3.3. Resource pooling

The provider's resources are pooled to serve consumers using a **multi-tenant** model, with different physical and virtual resources dynamically allocated according to consumer demand.

1. National Institute of Standards and Technology (NIST), U.S. Department of Commerce

Advantage for provider: efficiency in utilization

1.3.4. Rapid elasticity

Resources can be rapidly and elastically scaled up and down.
Advantage for consumers: quick and easy

1.3.5. Measured service

Pay as you go, pay only for what you use.

1.4. Why use cloud?

- Better capital utilization
- The unit cost of on-demand capacity may be higher than the unit cost of fixed capacity; offset by no charge when capacity is not being used
- Elasticity, easy to scale up and down
- Access to complex infrastructure and resources without internal resources.
- Providers: better resource utilization

1.5. Cloud deployment models

1.5.1. Public cloud

- Providers let clients access the cloud via Internet
- Made available to the general public
- Multi-tenant virtualization, global-scale infrastructure
- Functions and pricing vary

1.5.2. Private cloud

- The cloud is used solely by an organization
- May reside in-hourse or off-premise
- Secure, dedicated infrastructure with the benefits of on-demand provisioning

- Not burdened by network bandwidth and availability issues and security threats associated with public clouds.
- Greater control, security, and resilience.

1.5.3. Hybrid cloud

- Composed of multiple clouds(private, public, etc) that remain independent entities, but interoperate using standard or proprietary protocals
- Like Banks, hospitals, goverments
- Allows applications and data to flow across clouds

1.6. Issues of cloud

- Availability
- Data Loss
- Vendor lock-in (Apps built on a cloud may not be transferred to another)
- Networking
- Security
- Privacy

2. Service models

2.1. Outline

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)

2.2. IaaS

- Providers give you the computing infrastructure made available as a service.
- Providers manage a large pool of resources, and use virtualization to dynamically allocate.
- Customer "rent" these physical resoruces to customize their own infrastructure.

2.3. PaaS

- Providers give you a software platform, or middleware, where applications run.
- You develop and maintain and deploy your own software on top of the platform.
- The hardware needed for running the software is automatically managed by the platform. You can't explicitly ask for resources.
- You have automatic scalability, without having to respond to request load increase/decrease

2.4. SaaS

- Providers give you a piece of software/application. They take care of updating, and maintaining it.
- You simply use the software through the Internet

3. Virtualization

3.1. Outline

- Concepts
- Virtualization architecture
- CPU and OS basics
- Types of CPU virtualization
- Cloud infrastructures

3.2. Concepts

Allows one computer to "look like" multiple computers, doing multiple jobs, by sharing the resources of a single machine across multiple environments.

3.2.1. Some terms

- guest/VM
- guest OS
- host OS
- host

3.3. Virtualization architecture

3.3.1. Hosted architecture

A hosted architecture installs and runs the virtualization layer as an application on top of an operating system

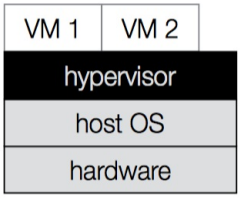


Figure 1.

Indirect access to hardware through the host OS
Performance penalty, usually for desktops and personal use

3.3.2. Hypervisor architecture

The hypervisor architecture installs the virtualization layer, called **hypervisor**, directly on a clean x86-based system

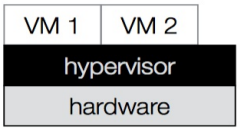


Figure 2.

It has direct access to hardware resources.

A hypervisor is more efficient than a hosted architecture and delivers greater scalability, robustness, and performance.

This architecture is for production use.

3.4. OS review

3.4.1. What is OS?

A special layer of software that provides application access to hardware resources

- Convenient abstraction of complex hardware devices
- Protected acess to shared resources

- Security and authentication
- Communication amonst logical entities

3.4.2. Four fundamental OS concepts

- Thread

  Single unqiue execution context.

  Program Counter, Registers, Execution flags, Stack.

- Address Space w/Translation
- Process
- Dual Mode operation/Protection

  User mode.

  Kernel mode.

3.4.3. Protection rings

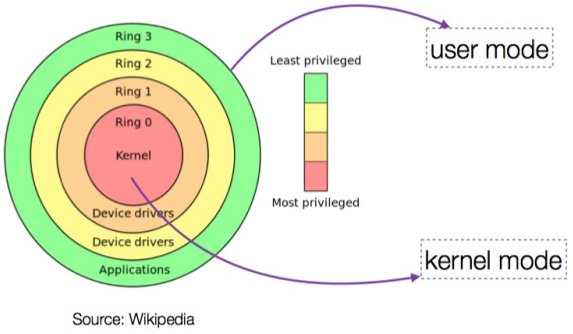


Figure 3.

3.5. CPU virtualization

Basically, the CPU does not care whether you are the guest OS or not
If it's unpriviledged code, code that execute in userspace
It's safe to run no matter it is from the guest OS or host OS

About priviledged code, there're 3 implementations currently.

- Full virualization
- Para virtualization
- Hardware-assisted virtaulization

3.5.1. Full virtualization

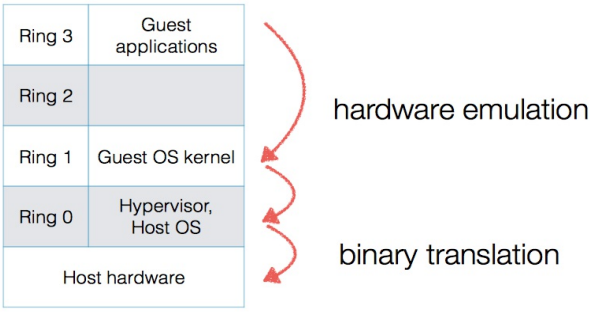


Figure 4.

Hardware is emulated by the hypervisor.

- The hypervisor presents a complete set of emulated hardware to the VM's guest operating system, including the CPU, motherboard, memory, disk, disk controller, and network cards.
- Regardless of the actual physical hardware on the host system, the emulated hardware remains the same.

Binary translation:

- Trapping I/O calls
- emulate/translate

The guest OS is tricked to think that it's running privileded code in **Ring 0**, while it's actually running in **Ring 1** of the host with the hypervisor emulating the hardware and trapping priviledged code

Unpriviledged instructions are directly executed on CPU.

Advantages:

- Keeps the guest OS unmodified
- Prevents an unstable VMs from impacting system performance; VM portability

Disadvantages:

- Performance is not good

3.5.2. Para-virtualization

This is developed to overcome the performance penalty of full virtualization with hardware emulation.

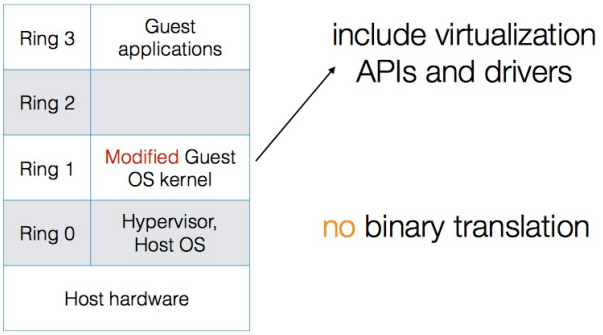


Figure 5.

Para-virtualization can be done in two ways:

- A recompiled OS kernel. Easy for Linux, Windows doesn't support.
- Paravirtualization drivers for some hardware, e.g. GPU, NIC

Guest OS is aware that it runs in a virtualized environment. It talks to the hypervisor through specialized APIs to run priviledged instructions.

These system calls, in the guest OS, are also called **hypercalls**

Performance is improved. The hypervisor can focus on isolating VMs and coordinating.

3.5.3. Hardware-assisted

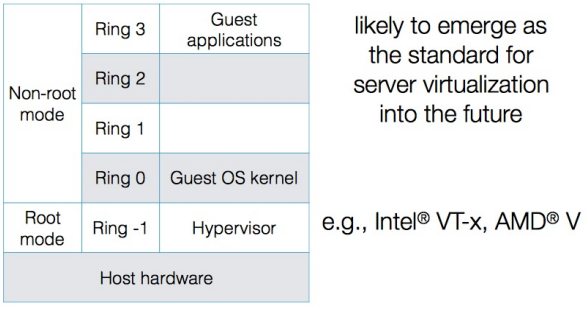


Figure 6.

Originally the machine is executing normally, without any guestOS. When the hypervisor launches a VM, an additional **Ring -1** layer for *Hypervisor* is added.

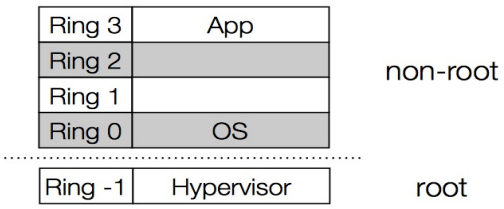


Figure 7.

When the guest OS meets certain triggers, which requires the hypervisor to exercise system control, a transition of control happens:

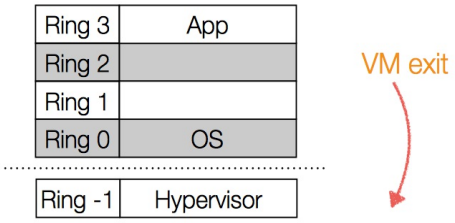


Figure 8.

When the hypervisor finished, the control switches back to non-root mode, the VM continues:

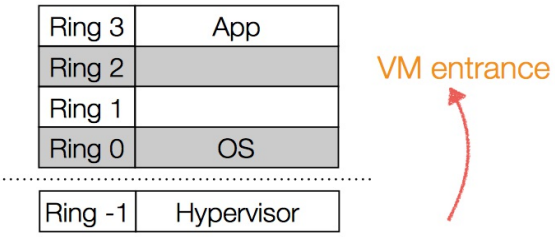


Figure 9.

The bigger picture is here:

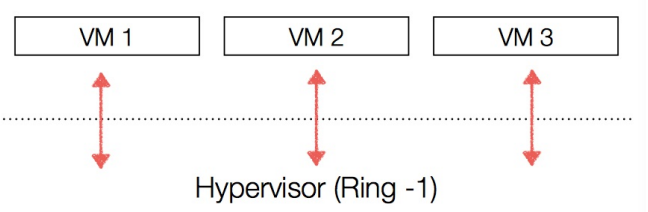


Figure 10.

Each VM is allocated a specfic hardware address space for performance isolation.

3.5.4. Summary

	Full	Para-	Hardware-assisted
Handling priviledged insturctions	binary translation	hypercalls	non-root/root mode
Guest OS modifications	No	Yes	No
Performance	Good	Best	Good
Examples	VMware, VirtualBox	Xen	Xen, VMware, VirtualBox, KVM

Table 1.

3.6. Cloud Infrastructures

Cloud computing is usually related to virtualization

- Because of elasticity
- Launching new VMs in a virtualizaed environment is cheap and fast

A cloud infrastructure is in fact a virtual machine management infrastructure.

4. Cluster Scheduling

4.1. Background

Rapid innovation in cluster computing frameworks

4.1.1. Trends Observerd

- Diverse workloads
- Increasing cluster sizes
- Growing job arrival rates

4.1.2. Workload: batch/service split

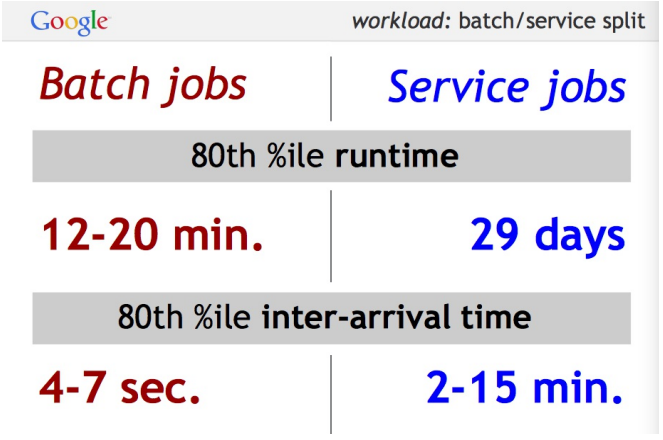


Figure 11.

4.2. Existing Approaches

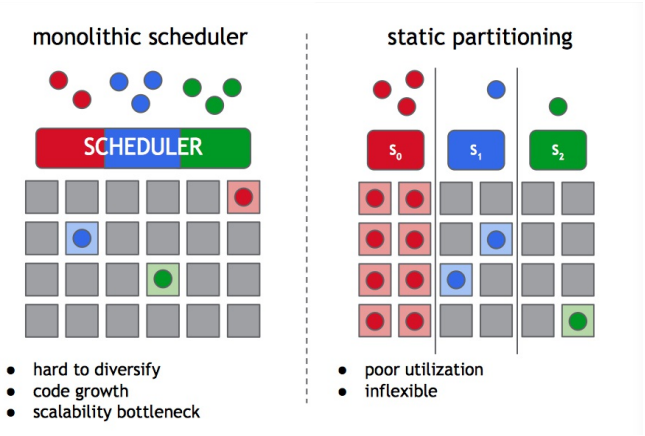


Figure 12.

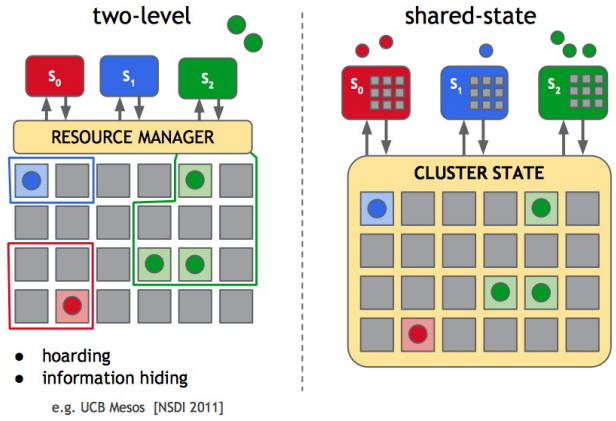


Figure 13.

4.3. Mesos

A platform for Fine-Grained Resource Sharing in the Data Center

Mesos is a common resource sharing layer over which diverse framework can run.

4.3.1. Mesos Goals

- **High utilzation** of resources
- **Support diverse frameworks**
- **Scalability** to 10, 000's of nodes
- **Reliability** in face of failures

4.3.2. Design Elements

Fine-grained sharing:

- Allocation at the level of tasks within a job
- Improves utilization, latency, and data locality

Resource offers:

- Simple, scalable application-controlled scheduling mechanism

4.4. Omega: shared-state

4.5. Job Scheduling Fairness

- n users want to share a resource
- Generalized by **max-min fairness**
- Generalized by **weighted max-min fairness**

4.5.1. Properties of max-min fairness

- Share guarantee

  Each user can get at least 1/n of the resource.

  But will get less if they demand is less.
- Strategy-proof

  Users are not better off by asking for more than they need.

4.5.2. Desirable properties of max-min fairness

- Isolation policy

  A user gets her fair share irrespective of the demands of other users.

- **Flexibility** separates mechanism from policy

  Proportional sharing, priority, reservation.

4.5.3. Fair share multiple resources for heterogenous demands

Users have *tasks* according to a *demand vector*.

e.g. <2,3,1> user's tasks need 2 R_1, $3R_2$, $1R_3$.

Resources given in multiples of demand vectors.

A natural Policy

Asset Fairness

Equalize each user's *sum of resource shares*.

Share Guarantee

Every user should get $1/n$ of at least one resource.

Strategy-proofness

A user should not be able to increase her allocation by lying about her demand vector.
Intuition:
Users are incentivized to provide truthful resource requirements

Dominant Resource Fairness

A user's **dominant resource** is the esource she has the biggest share of.
A user's **dominant share** is the fraction of the dominant resource she is allocated.

Apply max-min fairness to dominant shares.
Equalize the dominant share of the users.

5. CPU Scheduling

5.1. Outline

- Why need CPU schduling?
- Some definitions
- Three scheduling algorithms in *Xen*

5.2. Motivation

When multiple VMs send instructions to the host, which instructions should the host CPU execute first?

Resource contention

CPU scheduling is important in delivering performance guarantees and resource isolation.

5.3. Fairness

- Idea : use weights
- Each VM gets a share of CPU in proportion to the weight
- Weighted max-min; Proportional fairness, PS

5.4. Utilization

5.4.1. Work-conserving

The host CPU is idle if and only if there is no runnable VM. The weights act as guarantees. Utilization is good.

5.4.2. Non work-conserving

Weights are caps. Each VM owns its fraction of CPU.

Tradeoff between fairness and utilization

- work-conserving schedulers are more efficient, but less fair
- non work-conserving schedulers are less efficient, but more fair.

5.5. Three CPU Schedulers in Xen

5.5.1. Borrowed Virtual Time (BVT)

The concept is originally developed for processor scheduling in OS.

The unit of time is minimum charging unit (MCU). Typically the frequency of clock interrupts.

Each VM (i) has a weight (w_i), and a virtual time counter(v_i).

If a VM is scheduled to run, its virtual time is incremented:

$$v_i = v_i + \frac{t}{w_i}, t: \text{wall clocktime}$$

The scheduler dispatches a runnable VM with the smallest virtual time first.

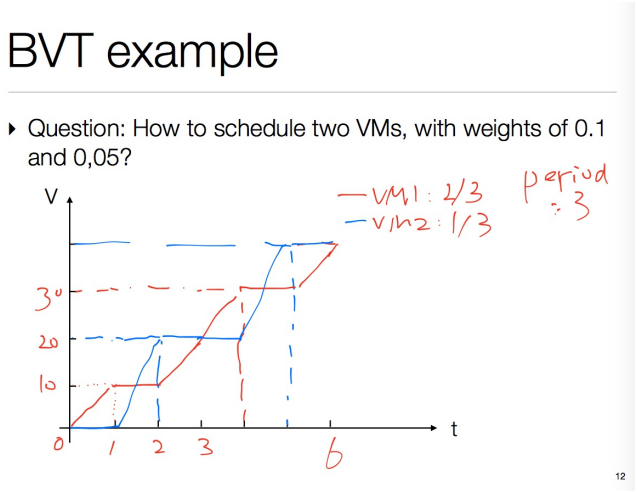


Figure 14.

BVT is work-conserving

A VM can warp back in virtual time, in order to gain priority over other VMs.

The warp, say 10 (MCUs), the virtual time starts from -10 instead of 0. This is the same as *borrowing* time from future.

Advantages: Good fairness, low overhead implementation on multiprocessors and uniprocessors.
Disadvantages: No non-work conserving support. This is important for performance guarantees.

5.5.2. Simple earliest deadline first (SEDF)

EDF is a widely used scheduling algorithm in OS.

Each VM specifies its CPU requirement with a tuple (s_i, p_i, x_i)

VM_i wants to receive at least s_i units of time in each period of length p_i.

x_i: whether to receive extra CPU time (work conserving)

For each VM, the scheduler keeps two variables:

- d_i: time at which VM_i's current period ends, i.e. *deadline*
- r_i: remaining CPU time in the current period

At each time slot, among the VMs whose r_i is positive, schedule the one with the earilest deadline to run. Ties are broken arbitrarily.

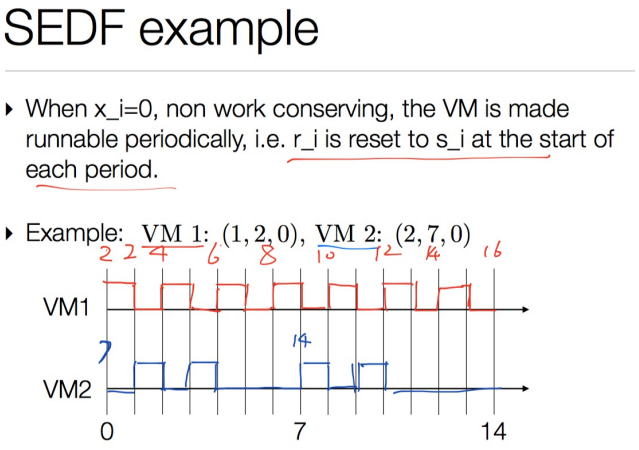


Figure 15.

The least common multiple of the periods is 14. So the scheduling pattern will repeat every 14 slots.

The CPU utilization of the above example is: $\frac{1}{2}+\frac{2}{7}=\frac{11}{14}$

Any VM set is schedulable by SEDF if and only (iff)

$$\sum \frac{s_i}{p_i} \leqslant 1$$

Run te scheduability test first before admitting a new VM

The **time granularity** of the period impacts scheduler fairness.

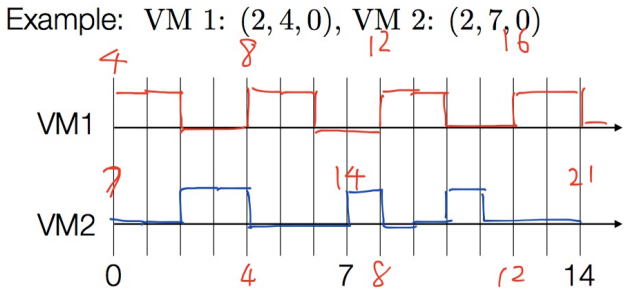


Figure 16.

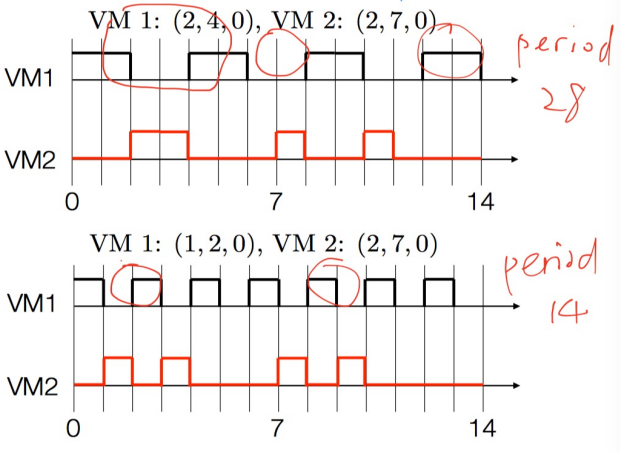


Figure 17.

In WC mode($x_i=1$), a VM is always runnable (given it has work to do). r_i is reset to s_i immediately after it reduces to 0, and d_i is incremented by p_i at the same time.

The slack CPU time is allocated to VMs (whose $x_i=1$) in a weighted fair sharing manner.

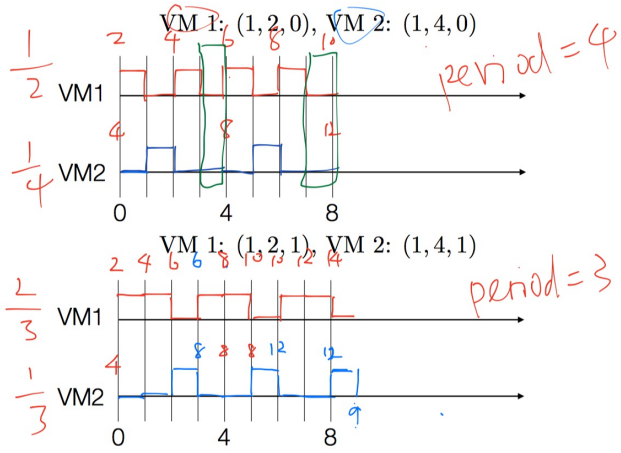


Figure 18.

Features:

- Fairness depends on the value of the period
- WC and NWC modes

Disadvantage:

- Implements per-CPU scheduling. Lacks global load balancing on multiprocessors.

5.5.3. Credit Scheduler

Xen's latest PS(proportional share) scheduler featuring automatic load balancing across physical CPUs.

Each VM is assigned a weight and a cap

Weight: default value is 256. range: [1, 65535]

Cap: optionally fix the CPU this VM can get, even if the CPU has idle cycles (NWC)

Default is 0, WC
Set to percentages. For example 30, meaning this VM can get 30% of the CPU at most.

Basics

Time unit/slot: 30ms

A VM has priority, which can be one of two values: “over” or “under” representing wether this VM has or hasn’t yet exceeded its fair share of CPU resource in the ongoing accounting period.

Each CPu manages two FIFO queue of runnable VMs sorted by priority (under precedes over)
At each slot, the VM off the head of the queue gets to run.

As a VM runs, it consumes credits. Every 30ms, a system0wide accounting thread recomputes the credits for each active VM.

Every so often, the system adds credit to each VM. The credit is allocated in a PS fashion (weighted fair sharing).

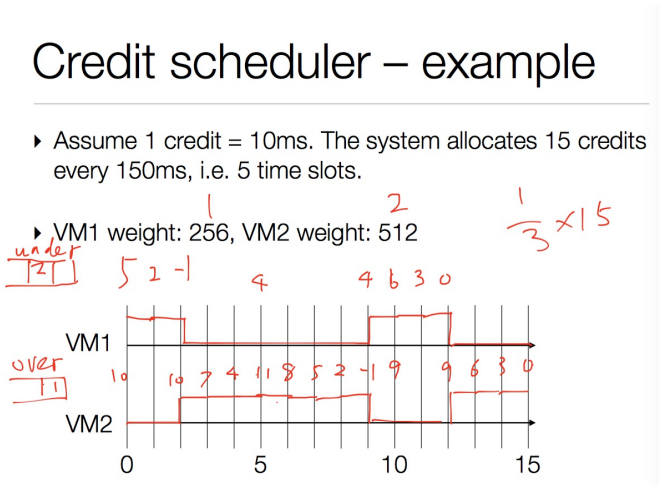


Figure 19.

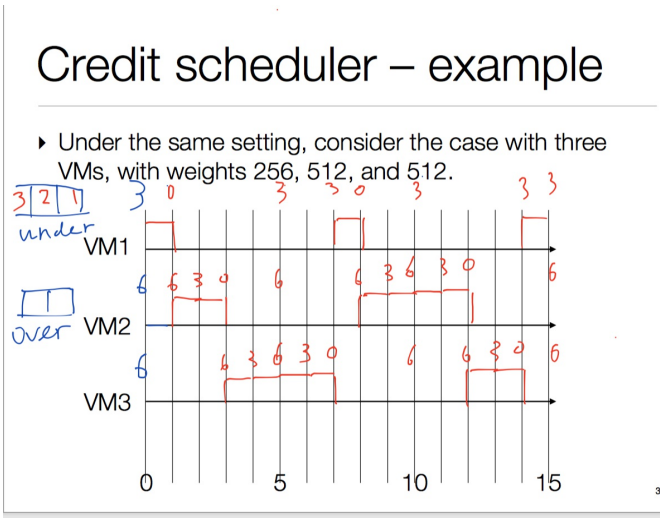


Figure 20.

The use of credit makes it easy to schedule VMs over multiple CPUs. Whichever CPU the VM is using, just update the credit.

When a CPU doesn't find a VM of priority under in its local queue, it will look on other CPUs for one. This guarantees that no CPU idles when there is runnable work in the system.

Features:

- Global load balancing
- WC and NWC modes
- Easy to implement

6. Data Center Network - Basics, Topology, Traffic

6.1. Outline

- Brief history of computer networking and the Internet
- Basic concepts
- Layering architecture
- TCP/IP

6.2. History

1. In 1973, R. Metcalfe invented Ethernet
2. In 1974, Vinton Cerf and Robert Kahn invented TCP/IP
3. The term "internet" was adopted around 70's as an abbreviation of *internetworking*
4. Kill applications: Email, Web, peer-to-peer (P2P), search engine, video, mobile, social media

6.3. Some basic concepts

6.3.1. Building blocks

Nodes

- end hosts, or hosts: PC, server
- switches/routers, middleboxes

Links: coax cable, optical fiber, wireless, ...

- point-to-point
- multiple access (shared)

6.3.2. Switching approaches

- Circuit switching
- Packet switching

6.3.3. Addressing, routing

- Address: byte-string that identifies a node
- Routing: process of forwarding messages to the destination node based on its address
- Types of addresses:

  unicast: node-specific.

  broadcast: all nodes on the network.

  multicast: a group/subset of nodes.

6.3.4. Multiplexing

- Time-division multiplexing (TDM)
- Frequency-division multiplexing (FDM)

6.3.5. Statistical multiplexing

- On-demand time-division. Schedule link on a per-packet basis
- Packets that content for the link enter a/some buffer(s)/queue(s)
- Different scheduling disciplines can be used to decide which packet to transmit next.
- Buffer (queue) overflow is called congestion.

statistical multiplexing gain:
not everybody is talking!

6.3.6. Performance metrics

- Bandwidth, throughtput

  amount of data transmitted per unit time.

  1 Mbps = 10^6 bits per second, 1MBps = 8 Mbps.
- Latency (delay)

  total time from one end to another.

  latency=propagation + transmission + queue.

  propagation = distance/C, transmission = size/bandwidth.

Relative importance

- 1B flow: queueing delay dominates

  1ms/100ms vs. 1MBps/100MBps.
- 25MB flow: transmission delay, i.e. throughput, dominates

  1ms/100ms vs. 1MBps/100MBps.

6.4. Layering architecture

6.4.1. Why layering?

- Functions are separated into layers. Layers are *blackboxes* to the upper- or lower-layer protocols.
- Allow distinct technologies for the same function

  physical: Ethernet, 3G, WiFi, satellite, optical etc.
- Allow them to inter-operate using standardized interfaces

6.4.2. OSI reference model

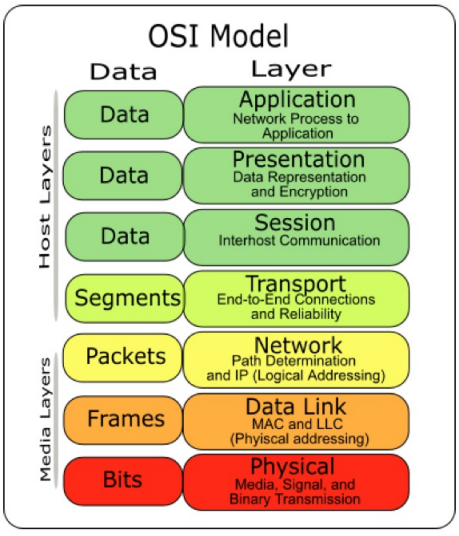


Figure 21.

6.4.3. More popular TCP/IP model

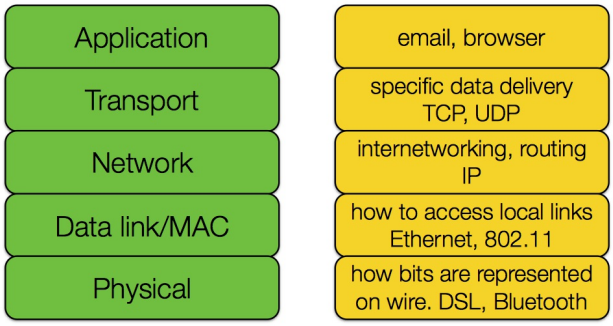


Figure 22.

6.5. TCP/IP

6.5.1. Internet protocol - IP

- Connectionless, packet-based
- Best-effort, inherently unreliable

  packets may be losts, dropped, delayed, deliverd out-of-order.

  no guarantee.
- Header format

6.5.2. Routing

- Every packet header has the destination address
- If the router is directly connected to the destination network, forward to the host
- else, forward to some other router

6.5.3. Transport layer

Provide **reliable** data transfer to applications over **unreliable** IP network

6.5.4. TCP overview

- TCP: Transmission Control Protocol
- Point-to-point; reliable in-order byte stream (sequence numbers); full duplex

- Connection-oriented, handshaking before data exchange
- Flow control: not sending too fast for the receiver to process (src and dot may have different network speeds). recv window, rwnd
- Congestion control: avoid congestion collapse (too many sources sending too much), Congestion window, cwnd

6.5.5. Reliable transfer

- Use sequence number, and send acknowledgement packets indicating the sequence number up to which the receiver has received.
- If some packets are received out-of-order, or are lost, the reciever will only ACK the last seq num of the contiguous stream.
- Packet loss can be detected by timeouts and duplicated ACKs

6.5.6. TCP flow control

- Receiver controls sender, so the sender won't overflow the receiver's buffer by sending too fast
- Receiver advertises buffer space by including a *rwnd* value in the header
- Sender limits the amount of un-acked data to receiver's *rwnd* value

6.5.7. TCP congestion control

Manifestations:

- lost packets (buffer overflow at routers)
- long delays (queueing in router buffers)

TCP relies on packet drops as a signal of congestion. Packet drops are detected by duplicated ACKs. Thus when TCP sees duplicated ACKs, it will interpret as congestion is experienced.

However,

1. Packet drops may not be caused by congestion, e.g. in wireless networks
2. Duplicated ACKs may not be caused by packet drops. They may be due to a change of network path and some packets arrive late bu not dropped.

Approach: AIMD

Sender increase sending rate(window size), probing for unused bandwidth, until loss occurs

- additive increase: increase cwnd by 1 MSS (maximum segment size) every RTT until loss detected
- multiplicative decrease: cut cwnd by half when loss is detected.

6.6. Topology

An evolution of Google's network topologies.

6.6.1. 2004: four-post cluster

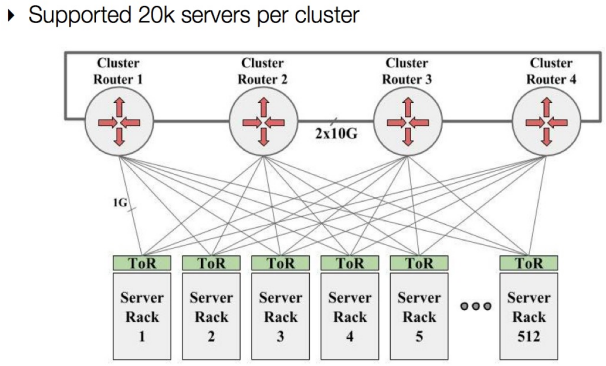


Figure 23.

But the traffic keeps growing, then change to Clos topologies.

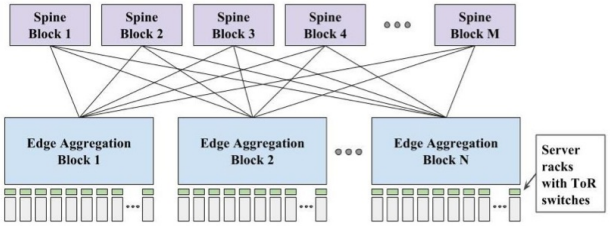
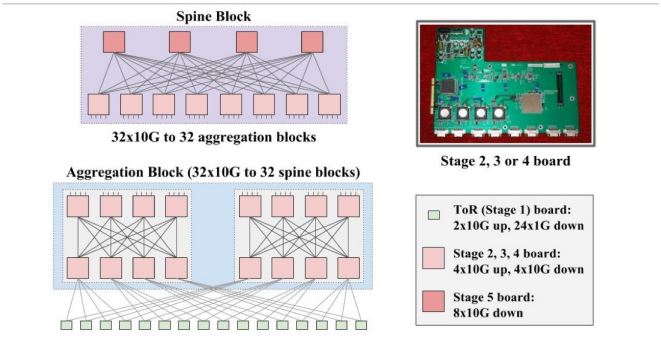


Figure 24.

Use Clos topologies

- Use many low-radix switches in multiple stages to scale to arbitrary size
- Substantial path diversity and redundancy

6.6.2. 2005: Firrehose 1.0



- An aggr block has 16 ToRs (320 machines). 32 spine blocks each connect to 32 aggr blocks, resulting in a fabric that scales to 10K machines with 1G average bandwidth

Figure 25.

6.6.3. 2012: Jupiter

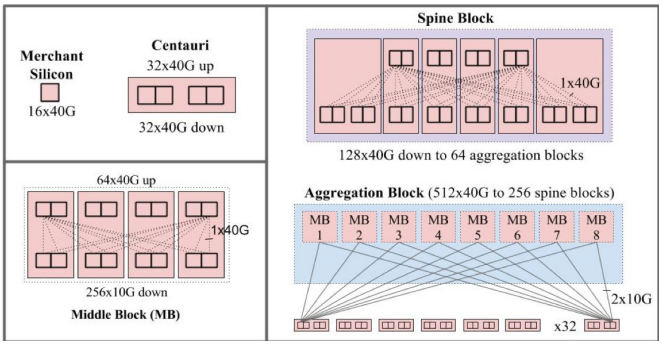
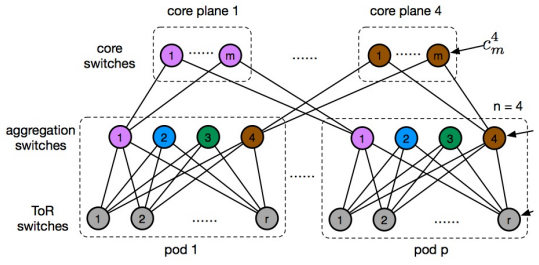


Figure 26.

6.6.4. Facebook's fabric



With 96 pods, the topology can accommodate 73,728 10Gbps hosts. In Facebook's Altoona data center, each aggregation switch connects to 48 ToR switches in its pod, and 12 out of the 48 possible core switches on its plane, resulting in a 4:1 oversubscription.

Figure 27.

6.6.5. Academia: Fat-tree

Clos topology, built upon commodity switches

k-pod fat-tree, k=4. There are k pods, each with two layers of $\frac{k}{2}$ switches.

Each aggr. switch has $\frac{k}{2}$ ports to unique core switches; $\frac{k}{2}$ aggr. switches in a pod. So total number of core switches is $\left(\frac{k}{2}\right)^2$.

Each core switch has one port to each pod.

Each edge switch has $\frac{k}{2}$ hosts. $\frac{k}{2}$ edge switches in each of k pods. So a k-pod fat-tree can support $\frac{k^3}{4}$ hosts.

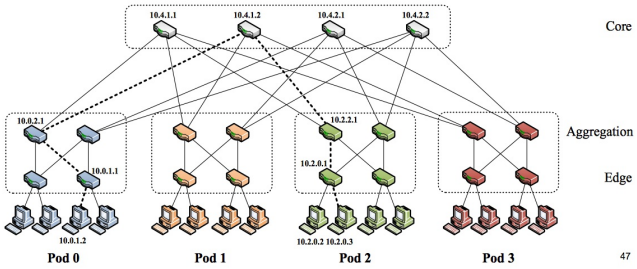


Figure 28.

Advantages of fat-tree

- In tranditional hierarchical networks, switches in aggregation and core layers need to be more powerful, and have more ports per device. High-end high port density switches are extremely expensive.
- Scale out vs. scale up
- Fat-tree: $\left(\frac{5k^2}{4}\right)$ k-port switches support $\frac{k^3}{4}$ hosts.
- 48-port 1GigE switches: 27,648 hosts using 2,880 switches.
- Rearrangeably non-blocking: for arbitary communication patterns, there is some set of paths that will saturate all the bandwidth available to the end hosts in the topology.
- Hierarchical topologies are not rearrangeably non-blocking.

6.7. Traffic characteristics

6.7.1. Flow size

Mice vs. elephants: many mice flows, a few elephants carrying most of the bytes -> Routing elephants is important

6.7.2. Localility

Mostly cluster-local; more traffic cross data center than within the rack -> inter-DC WAN is important

Hadoop generates the most traffic

7. Data Center Networking - Latency

7.1. Outline

- The tail latency problem
- Solutions

7.2. Multipath routing

7.2.1. Multipath in Fat-tree

In general, there are multiple shortest paths from one host to another (in a different pod). These shortest paths have the same hop distance - equal cost

So, how to choose a pth out of all equal-cost ones?

The routing decision is done at each switch locally. In fat-tree, all upstream links of a switch can be used to reach a host in another pod.

Sate-of-the-art: ECMP, equal cost multipathing.

7.2.2. ECMP

- Consistent hashing of five tuples -> packets of the same flow go through the same path.
- Hashing uniformly spreads flows across all links. Each port has an equal number of flows on average.
- Easy to implement in hardware. No need to maintain per-flow state in switches.

7.3. Latency problem

7.3.1. Traffic characteristics

- >80% of flows are mice that are less than 100KB. >90% of all bytes are from elephants however.
- Elephants: MapReduce, data mining, indexing. Mice: query/response applications, Web
- The mixed nature causes unbalanced traffic with ECMP.

ECMP routes traffic on a per flow basis. Due to the mixed nature of traffic, hash collisions can happen, and ECMP cannot uniformly balance the load across network.

Mice flows may suffer head-of-line blocking in switches, leading to long latency (elephants consume all buffer)

Tail latency, i.e. latency at 99-th or 99.9-th percentile, can be much worse than average latency.

For many applications, the final response time depends on the slowest flow. The tail latency, not the average latency, determines the application response time.
It is crucial to reduce tail latency in data center networks.

7.4. Solutions

7.4.1. Overview

- Reduce queue length at switches[2]
- Prioritize mice flows

  Switches should always schedule mice flows' packets first before elephants.
- Better multipath routing

  Leads to less hash collisions.

7.4.2. Reduce queue length: DCTCP

2. TCP uses packet drops as a conestion single. It's too late: we can throttle the elephant before they use up the buffer

Uses ECN, Explicit Congestion Notification, commonly available at switches.

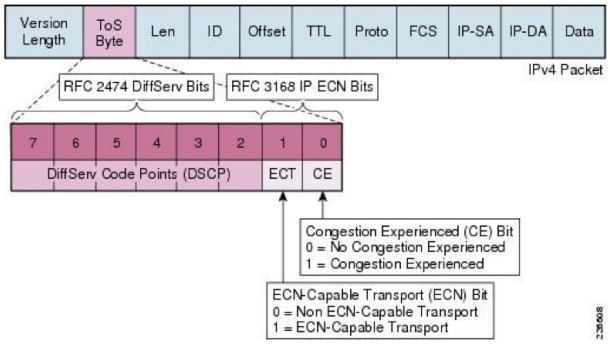


Figure 29.

Whenever the queue length exceeds a small threhold, switches set the CE bit in IP header.

Then the receiver sets the ECN-echo flag in the tCP header of ACKs

The sender gets ECN information from the ACKs, and performs congestion control to throttle(限流) its sending rate.

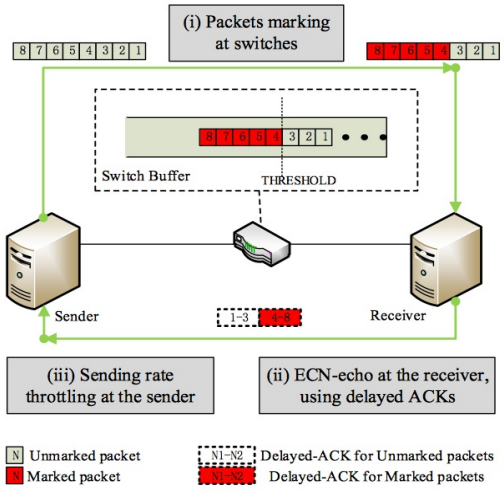


Figure 30.

The sender maintains an estimate of the fraction of packets that are marked for each congestion window, say α. α is updated based on the old value from the previous window, and the received ACKs in the current window.

Upon getting an ECN marked ACK, the senders does the following congestion control:

$$\text{cwnd} \leftarrow \text{cwnd} \times \left(1 - \frac{\alpha}{2}\right)$$

DCTCP's congestion control is quicker, with a finer granularity, compared to TCP.

7.4.3. Prioritize mice flows: pFabric

- pFabric: a clean-slate, minimalist network fabric, dramatically different from current data center network design.
- idea: decouple flow scheduling from congestion control
- Switches: prioritize flows. Hosts: no congestion control, since we have flow prioritization.

Each packet encodes a priority number in its header.
Hosts: send all packets at line rate.
Switches: priority scheduling, priority dropping

7.4.4. Better multipath routing: RepFlow

- Observation: Hash collisions and congestion happen randomly in any part of the network
- Idea: from all the equal-cost paths, not every one of them has long latency -> replicate each mice flow, and send them via different paths.
- Just launch two network sockets, with different src port numbers. This also makes sure that ECMP will hash the original and replicated flow to different paths.
- **Advantage**: no change to kernel/switches. Works for any transport protocols.

7.4.5. Congestion-aware LB

Fundamentally improve LB by making it *congestion-aware*.

Modify switch data plane

- Monitor link utilization
- For a new flow (flowlet/flowcell), collect path load for all/some of the equal-cost paths, make a LB decision.

7.4.6. A strawman design

A naive path-wisse feedback approach. [3]

3. tracks end-to-end congestion levels of all paths

works only for *2-tier* Clos(i.e. leaf-spine), *not scalable* for 3-tier Clos.

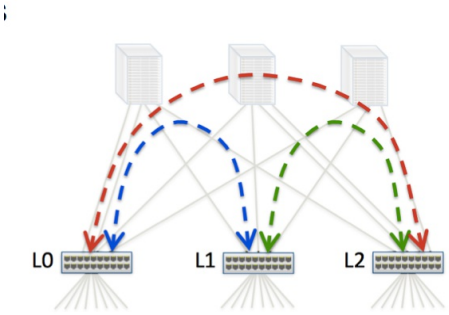


Figure 31.

8. Data Center Networking - QoS

8.1. Outline

- Queueing disciplines
- Traffic shaping
- Rate delay guarantees

8.2. Queueing disciplines

Each router must implement some queueing discipline, including scheduling, and drop policy.

Queueing allocates both bandwidth and buffer space

- Bandwidth: scheduling, which packets to serve next
- Buffer: which packet to drop

Queueing also affects latency

8.2.1. FIFO Queueing

FIFO+DropTail queue:

- Most widely used
- Queue packets First In First Out
- Drop packets at the tail when buffer is full

Bandwidth sharing with FIFO queueing
Bandwidth allocation depends on behavior of all flows.

Aggressive user/flow can crowd out the others

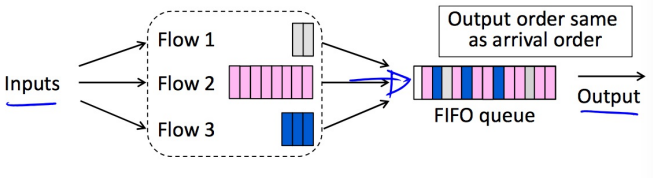


Figure 32.

8.2.2. Round-Robin Queueing

To improve fairness of sharing bandwidth.

Queue packets separately for each flow; take one packet in turn from each non-empty flow at next output time.

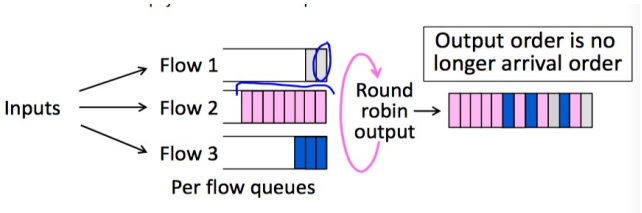


Figure 33.

Flows don't see uncontrolled delay/loss from others. But different packet sizes lead to bandwidth imbalance.

8.2.3. Fair Queueing

Fair queueing is like Round-Robin but approximate bit-level fairness, by computing virtual finish times.

- Virtual clock ticks once for each byte sent from all flows.
- Send packets in order of their virtual finish times, $\text{Finish}(j)_F$
- Not perfect - don't preempt packet being transmitted.

$$\text{Arrive}(j)_F \quad \text{arrival time of } j-\text{th packet of flow } F$$

$$\text{Length}(j)_F \, \text{length / bytes of } j-\text{th packet of flow } F$$

$$\text{Finish}(j)_F = \max\left(\text{Arrive}(j)_F, \text{Finish}(j-1)_F\right) + \text{Length}(j)_F$$

Example
Flow 1 and 3 use 1000B packets, while flow 2 uses 300B packets.
What will fair queueing do?

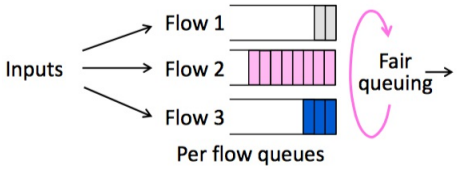


Figure 34.

Let $\text{Finish}(0)_F = 0$, queue backlogged $[\text{Arrive}(j)_F < \text{Finish}(j-1)_F]$
$\text{Finish}(1)_{F_1} = 1000, \text{Finish}(2)_{F_1} = 2000, \ldots.$
$\text{Finish}(1)_{F_2} = 300, \text{Finish}(2)_{F_2} = 600, \text{Finish}(3)_{F_2} = 900, 1200, 1500, \ldots.$
$\text{Finish}(1)_{F_3} = 1000, \text{Finish}(2)_{F_3} = 2000, \ldots.$

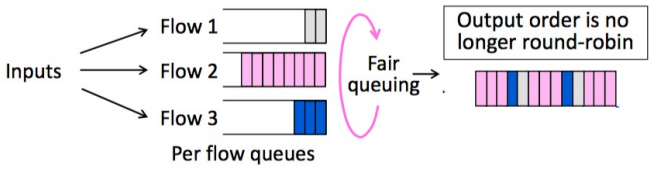


Figure 35.

8.2.4. Weighted Fair Queueing (WFQ)

WFQ is a useful generalizaiton for Fair Queueing:

- Assign a weight, Weight_F, to each flow
- Higher weight gives more bandwidth proportionally
- Change computation of $\text{Finish}(0)_F$ to factor in the weight

$$\text{Finish}(j)_F = \max\left(\text{Arrive}(j)_F, \text{Finish}(j-1)_F\right) + \frac{\text{Length}(j)_F}{\text{Weight}_F}$$

Potentials

- Can prioritize and protect flows
- A powerful building block

Not yet complete solution

- Need to determine flows(user?application?TCP flow?)
- Difficult to implement at high speed for many concurrent flows
- Need to assign weights to flows

8.3. Traffic shaping

8.3.1. Motivation

Shaping traffic flows constrains the load they put on the network

- Limiting the total traffic enables bandwidth guarantees
- Limiting bursts avoids unnecessary delay and loss

To describe traffic flows to the network, **average** rate mattersl relates to long-term bandwidth. **Burstness** also matters; relates to short-term bandwidth

The figure should be 1. more expressive than average 2. still relatively simple.

8.3.2. Token bucket

(R, B) token bucket constrains:

- average rate of R bits/sec
- bursts (over R) of B bits

Shaping modifies traffic near the source to fit within an (R, B) profile.

- Run a (R, B) token bucket at the source
- Pass sent packets to the network when tokens are available
- Delay (queue) packets while more tokens arrive

Let users condition their traffic to meet the network requirements

Token buckets help the user and network regulate traffic for QoS

Special treatment is implemented with other means such as WFQ at the network

8.4. Rate delay guarantees

8.4.1. Motivation

Sometimes we want to set minimum rate and maximum delay regardless of how other flows behave.

8.4.2. Admission control

Suppose we have a flow F that needs rate $\geqslant R$ Mbps and delay $\leqslant D$ secs

We must decide whether to admit or reject it from the network - admission control

Key point is we need the ability to control load to make guarantees.

8.4.3. Single router rate guarantee

- WFQ can guarantee rate at a router.
- Consider flow F with weight 10. Suppose the total weights of all flows is 100.

  Flow F gets $\geqslant(\frac{10}{100}) \times 100 = 10$**Mbps.**

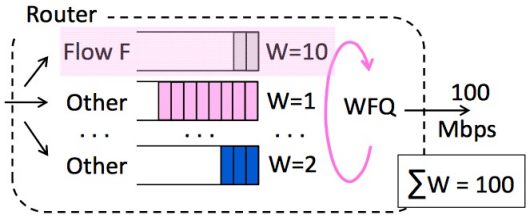


Figure 36.

8.4.4. Network rate guarantee

We can guarantee a minimum rate for a network path by guaranteeing it at each router.

Condition at each router is:
For all router i:
$\frac{W_F(i)}{W(i)} \times L(i) \geqslant R$ Mbps

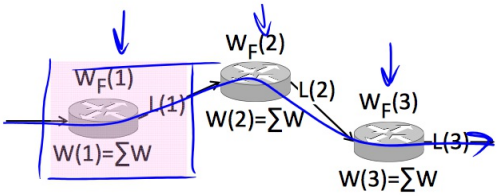


Figure 37.

8.4.5. Router delay guarantee

Assume the flow F is shaped by a (R, B) token bucket.
WFQ is used at the router with weights set such that F's rate $\geqslant R$ Mbps

In worse case, the maximum number of bytes that can move from the packet buffer to the queue during a small interval $[u, t]$ is bounded by $B + R(t - u)$.

By WFQ, the router guarantees a rate of at least R for our sender.

Thus, the maximum queueing delay seend by any packet of this flow at any time is bounded by $\boldsymbol{\frac{B}{R}}$

8.4.6. Network delay guarantee

If packets are smoothed out perfectly at rate R, then queueing delay is zero.

- Packets enter router just in time to leave.
- Even if the flow pays for a burst somewhere, the output packets are smoothed out at rate R for the next hop.

Therefore, we only pay for the burst only once.

Delay across N routers: $\leqslant$network delay $+ \frac{B}{R}$

8.4.7. Rate delay guarantees

Given a network with:

- (R, B) shaped traffic flow
- WFQ routers with proper weights for R
- Sharing with statistical multiplexing

We can guarantee the flow a minimum rate and maximum delay

- Rate is $\geqslant R$
- Delay is $\leqslant$network delay $+ \frac{E}{R}$

9. Storage Systems

9.1. Outline

- NoSQL storage systems - Distributed Storage Systems
- CAP theorem
- Amazon Dynamo
- Chord - Distributed hash table

9.2. NoSQL

Key-Value Stores	Column-oriented DB	Document DB	Others
Amazon Dynamo	Google Bigtable	MongoDB	Graph DB
LinkedIn Voldemort	Facebook HBase		memcached

Table 2.

9.3. CAP theorem

A distributed system cannot provide all three of the following properties at the same time:

- **C**onsistency: A read sees the result after all previously completed writes
- **A**vailability: Reads and writes always succeed, as long as the nodes do not fail
- **P**artition tolerance: Guaranteed properties still hold even when network failures prevent some nodes from communicating

CA is not a cohoerent option.

The modern interpretation: during a network partition, a distributed system must choose either Consistency or Availability.

9.3.1. Distributed data store

Usually favor **A** over C
How to write/read in a distributed system?

- Load balancing, server joining/leaving, failures

- A write request will be served by one server.
- A read request will be served by one server.

Basic problem statement: given a set of storage nodes, we need to store a set of objects on them, and retrieve them upon request.

Idea: have a consistent hashing function h(x). It maps an object to a number i, and stores it on node i.

Design objectvies:
Deterministic, efficient, load balancing

9.4. Chord

Chord is a distributed hash table widely used in distributed systems.

The consistent hash function assigns each node and key (represents an object) an m-bit identifier, say using SHA-256:

$$h(x) = \text{SHA256}(x) \text{mod} 2^m$$

Identifiers are ordered in an identifier circle modulo 2^m
Key k is assigned to the first node whose identifier is equal to or follows k in the circle. This node called the successor node of key k, denoted by *successor(k)*.

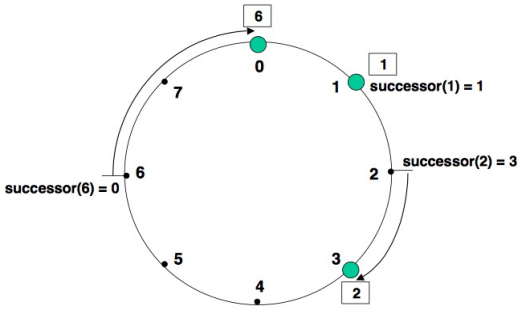


Figure 38.

A client connects to only one node, which probably isn't the successor of the object in question. There must be a distributed/scalable way for any node to figure out the successor of any key, to perform store/retrieve.

9.4.1. Solution 1

Each node only knows its successor node on the circle. Queries are passed around until they first encounter a node that succeeds the identifier.

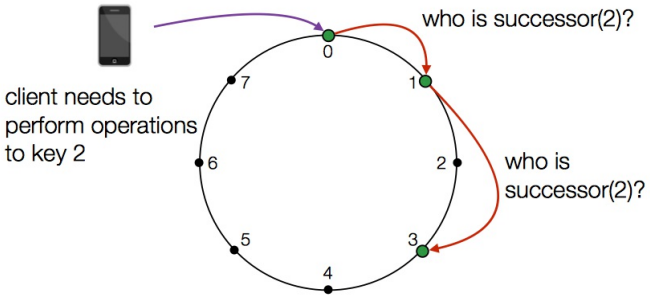


Figure 39.

Solution1 requires traversing all N nodes to find the appropriate mapping. The time complexity is $O(n)$.

9.4.2. Solution2 - finger table

Solution 2 stores additional info at each node to accelerate the location lookup.

Each node **n** will maintain a finger table with **m** entries.

The **i-th** entry at node **n** contains the identity of $\boldsymbol{s = \text{successor}(n + 2^{i-1})}$, its IP address and port number.

Node s is called the i-th finger of node n.

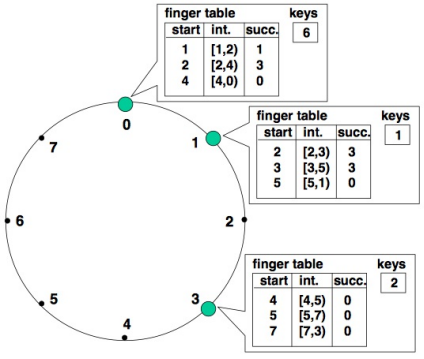


Figure 40.

- Each node only knows about m other nodes, and knows more about nodes closely following it than those farther away.
- Each node doesn't have enough info to determine the successor of an arbitrary key.
- What if a node doesn't know the successor of a key k? If it can find a node whose ID is closer than its own to k, that node will know more about the region of k.
- Node n searches its finger table for the node j whose ID most immediately precedes k, and asks j for the node it knows whose ID is closest to k.

Lookup algorithm

```
//ask node n to find id's successor
n.find_successor(id)
  n' = find_predecessor(id);
  return n'.successor;

//ask node to find id's predecessor
n.find_predecessor(id)
  n'=n;
  while(id ∉(n', n'.successor])
      n' = n'.closest_preceding_finger(id);
  return n';

//return closest finger preceding id
n.closest_preceding_finger(id)
  for i = m downto 1
      if(finger[i].node ∈ (n, id))
          return finger[i].node;
  return n;
```

Node joins/departures can also be handled gracefully by Chord(redistributing the keys, and reestablishing the finger tables).

9.5. Amazon Dynamo

Amazon engineers know that they can only have two properties for their DB and they choose **AP**.

Instead of strict consistency, Dynamo provides eventual consistency.

9.5.1. Node Failures

Dynamo deal with node failure by **replication**

Each object is stored first in successor(id), then the successor of successor(id), and so on, for a total of usually 3 copies.

9.5.2. Consistency

- Strong consistency: After a write reports a finished update, all subsequent reads return the updated value.

- Eventual consistency: After a write report a finished update, subsequent reads may report the old value. Eventually, a read will report the udpated value.

N = number of replicas for data
W = number of replicas that need to acknowledge the receipt of a write
R = number of replicas that are contacted for a read

$$\textbf{if } \boldsymbol{W + R > N}\textbf{, it is strong consistency}$$

$$\textbf{if } \boldsymbol{W + R \leqslant N}\textbf{, it is eventual consistency}$$

The common (N, R, W) setting in Amazon is $(3, 2, 2)$

10. Security

10.1. Outline

- Principles of security
- Plain Text and Cipher Text
- Encryption/Decryption

  Symmetric Key Cryptography.

  Asymmetric Key Cryptography.

10.2. Principles of security

- Confidentiality
- Integrity
- Authentication
- Non-repudiation

10.3. Plain Text and Cipher Text

10.3.1. Tranditional ways

- Substitution

  Caesar Cipher.

- Transposition

  Rail Fence Technique.

10.3.2. Some concepts

- Cryptography: a mechanism of encoding messages so that they can be sent securely
- Brute-force attack: try all combinations and permutations to decipher a message
- Encryption: encoding the plain text into cipher text
- Decryption: the reverse process of encryption

Encryption/decryption involves two aspects: **algorithm**, and **key**.

10.4. Cryptography

- Symmetric key cryptography

  The same key is used for encryption and decryption.

- Asymmetric key cryptography

  Two different keys are used.

10.4.1. Symmetric key cryptography

Communication channel is insecure. So how to set the key for encryption over insecure channel?

Diffie-Hellman key exchange
p, g: prime numbers.
a, b: random numbers.

Alice computes $s = g^{\mathrm{ba}} \bmod p$
Bob computes $s = g^{\mathrm{ab}} \bmod p$

The s is the secure key.

DES
Data Encryption Algorithm.

DES is a block cipher. It encrypts data in blocks of 64 bits. The key is 56 bits.

AES
Advanced Encryption Standard

10.4.2. Asymmetric key cryptography

- Pulic key: used for encryption
- Private key: used for decryption

RSA
The most popular and proven asymmetric key cryptography algorithm

It relies on a mathematical fact that it's easy to find and multiple two large prime numbers, but it's extremely difficult to factor their product back into two primes.

RSA key gen

- Choose two large prime numbers P and Q. (P=7, Q=17)

- Calculate N = P*Q=119
- Select the pulic key E such that it is not a factor of (P-1)(Q-1)=6*16. Ley's choose E=5
- Select the private key D such that the following is true:

  (D*E)mod(P-1)(Q-1)=1.
- Let's choose D=77, because 77*5 mod 96=1

RSA En/decryption

- Suppose the keys are generated by Bob. Bob gives Alice its public key E and the number N.
- Alice wants to send a character "F" to Bob. She'll use Bob's pulic key to encrypt it.

  $\mathbf{CT = PT^{E} modN = PT^{E} modP}$ ***Q*.**

  Alice sends 6^{5}mod119 = 41.
- Bob uses the following: $\mathrm{PT} = \mathrm{CT}^{D} \mathrm{modN}$

  Bob gets $41^{77} \mathrm{mod}\, 119 = 6$

Digital signatures

- A uses his private key to encrypt a message
- Anyone can check the message is signed by A by using A's public key
- Only A can sign the message

Digital certificates
Certificates: bind Bob's ID to his PK

Public key infrastructure - PKI

- Everyone must know the public key of root certificate authority(CA)
- Root CA can sign certificates
- Certificates identify others, including other authorities.