Distributed Computing Systems

Overview of Distributed Systems

Andrew Tanenbaum and Marten van Steen, *Distributed Systems – Principles and Paradigms*, Prentice Hall, c2002.

The Rise of Distributed Systems

- Computer hardware prices falling, power increasing
  - If cars did same, Rolls Royce would cost 1 dollar and get 1 billion miles per gallon (with 200 page manual to open door)
- Network connectivity increasing
  - Everyone is connected with “fat” pipes, even when moving
- It is *easy* to connect hardware together
  - Layered abstractions have worked very well
- Definition: a *distributed system* is
  “*A collection of independent computers that appears to its users as a single coherent system*”

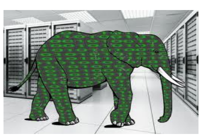
Why Distributed Systems?

A. Big data continues to grow:

- In mid-2010, information universe 1.2 zettabytes
- 2020 predictions 44x more at 35 zettabytes

B. Applications are becoming *data-intensive*.

- Big data - large pools of data captured, communicated, aggregated, stored, and analyzed
- Google processes 20 petabytes of data per day
- E.g., data-intensive app: astronomical data parsing

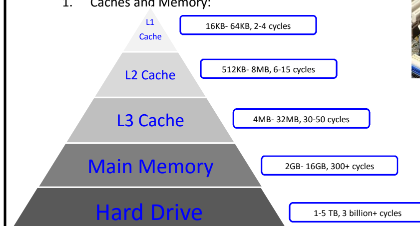


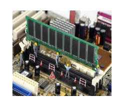
Ying Lu, UNL, *CSCE990 Advanced Distributed Systems Seminar*
http://cse.unl.edu/~ylu/csce990/notes/introduction.ppt

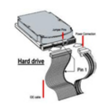
Why Distributed Systems?

C. Individual computers have limited resources compared to scale of current problems & application domains:

1. Caches and Memory:



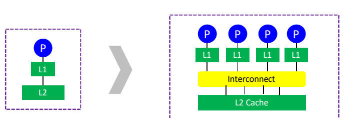




Why Distributed Systems?

2. Processor:

- Number of transistors integrated on single die has continued to grow at Moore’s pace
- Chip Multiprocessors (*CMPs*) are now available



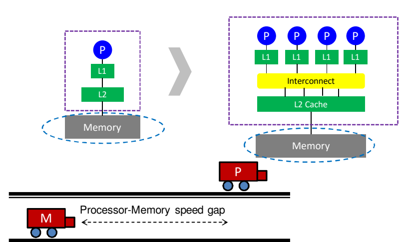
A single Processor Chip

A CMP

Why Distributed Systems?

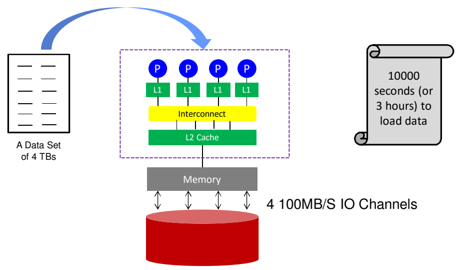
3. Processor (continued):

- CPU speed grows at rate of 55% annually, but mem speed grew only 7%

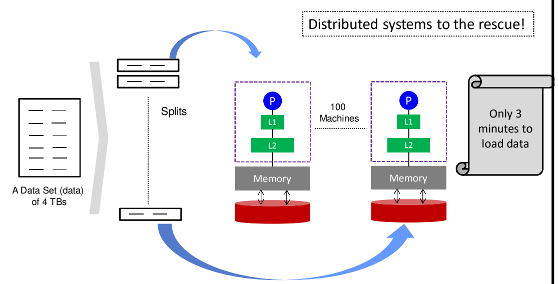


Why Distributed Systems?

- Even if 100s or 1000s of cores are placed on CMP, challenge to deliver stored data to cores fast enough for processing



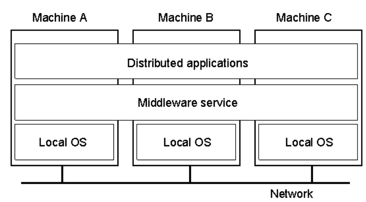
Why Distributed Systems?



But this brings new requirements

- A way to express problem as parallel processes and execute them on different machines (Programming Models and Concurrency).
- A way for processes on different machines to exchange information (Communication).
- A way for processes to cooperate with one another and agree on shared values (Synchronization).
- A way to enhance reliability and improve performance (Consistency and Replication).
- A way to recover from partial failures (Fault Tolerance).
- A way to protect communication and ensure that process gets only those access rights it is entitled to (Security).
- A way to extend interfaces so as to mimic behavior of another system, reduce diversity of platforms, and provide high degree of portability and flexibility (Virtualization)

Depiction of a Distributed System



Examples:
- The Web
- Processor pool
- Shared memory pool
- Airline reservation
- Network game
- The Cloud

- Distributed system organized as middleware. Note middleware layer extends over multiple machines.
- Users can interact with system in consistent way, regardless of where interaction takes place (e.g., RPC, `memcached`, …
- Note: Middleware may be "part" of application in practice

Outline

- Overview (done)
- Goals (next)
- Software
- Client Server
- The Cloud

Goal - Transparency

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be copied
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

(Different forms of transparency in distributed system)

Goal - Scalability

- As systems grow, centralized solutions are limited
  - Consider LAN name resolution (ARP) vs. WAN

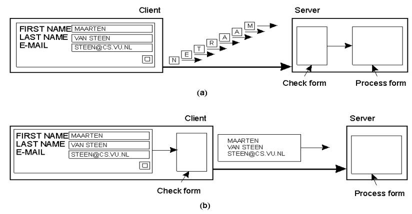
Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

- Ideally, collect information in distributed fashion and distribute in distributed fashion
- But sometimes, hard to avoid (e.g., consider money in bank)
- Challenges: geography, ownership domains, time synchronization
- Scaling techniques? → Hiding latency, distribution, replication (next)

Scaling Technique: Hiding Communication Latency

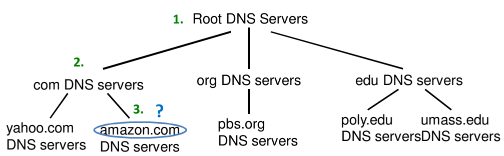
- Especially important for interactive applications
- If possible, do *asynchronous communication* – continue working so user does not notice delay

  - Not always possible when client has nothing to do
- Instead, can hide latencies



Scaling Technique: Distribution

- Spread information/processing to more than one location



Client wants IP for www.amazon.com (*approximation*):

1. Client queries root server to find `.com` DNS server
2. Client queries `.com` DNS server to get `amazon.com` DNS server
3. Client queries `amazon.com` DNS server to get IP address for `www.amazon.com`

Scaling Technique: Replication

- Copy of information to increase availability and decrease centralized load
  - Example: File caching is replication decision made by client
  - Example: CDNs (e.g., *Akamai*) for Web
  - Example: P2P networks (e.g., *BitTorrent*) distribute copies uniformly or in proportion to use
- Issue: Consistency of replicated information
  - Example: Web browser cache or NFS cache – how to tell it is out of date?

Outline

- Overview (done)
- Goals (done)
- Software (next)
- Client Server
- The Cloud

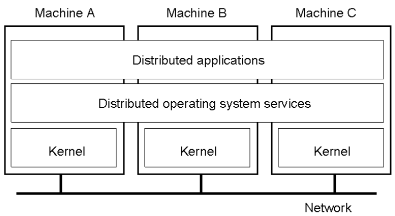
Software Concepts

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

(Next slides)

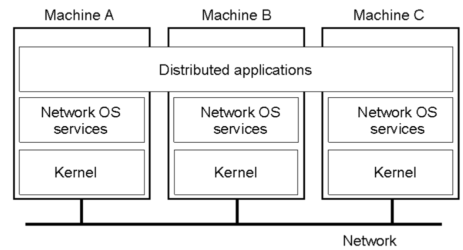
Distributed Operating Systems



- Typically, all hosts are homogenous
- But no longer have shared memory
  - Can try to provide *distributed shared memory*
    - But tough to get acceptable performance, especially for large requests

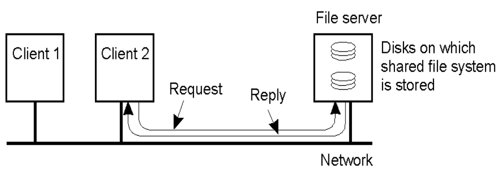
→ Provide *message passing*

Network Operating System (1 of 3)



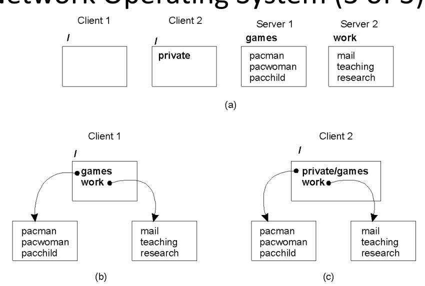
- OSes can be different (Windows or Linux)
- Typical services: `rlogin`, `rcp`
  - Fairly primitive way to share files

Network Operating System (2 of 3)



- Can have one computer provide files transparently for others (NFS)

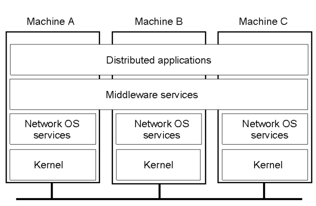
Network Operating System (3 of 3)



- Different clients may mount the servers in different places
- Inconsistencies in view make NOSes harder for users than DOSes
  - But easier to scale by adding computers

Positioning Middleware

- Network OS not transparent. Distributed OS not independent of computers.
  - Middleware can help



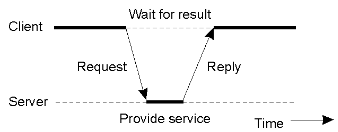
- Often middleware built in-house to help use networked operating systems (distributed transactions, better communication, RPC)
  - Unfortunately, many different standards

Outline

- Overview (done)
- Goals (done)
- Software (done)
- Client Server (next)
- The Cloud

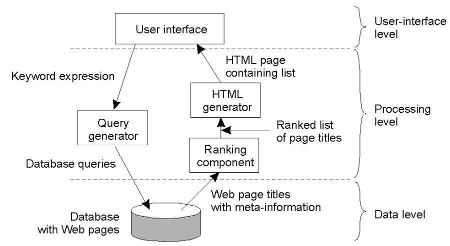
Clients and Servers

- Thus far, have not talked about organization of processes
  - Again, many choices but most widely used is *client-server*



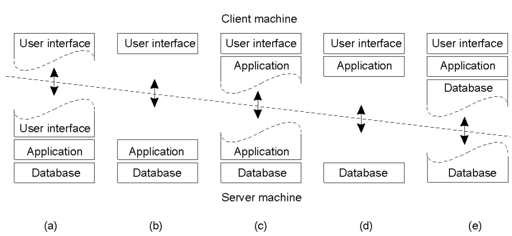
- If can do so without connection (local), quite simple
  - If underlying connection is unreliable, not trivial
  - Resend. What if receive twice?
- Use TCP for reliable connection (most Internet apps)
  - Not always needed for high-speed LAN connection
  - Not always appropriate for interactive applications (e.g., games)

Client-Server Implementation Levels



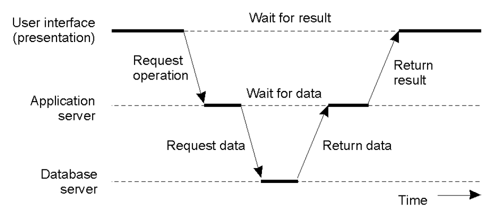
- Example of Internet search engine
  - UI on client
  - Data level is server, keeps consistency
  - Processing can be on client *or* server

Multitiered Architectures



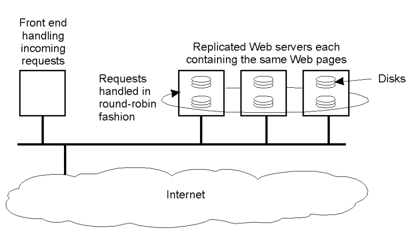
- Thin client (a) to Fat client (e)
  (a) is simple echo terminal, (b) has GUI at client
  (c) has user side processing (e.g., check Web form for consistency)
  (d) and (e) popular for NOS environments (e.g., server has files only)

Multitiered Architectures: 3 tiers



- Server(s) may act as client(s), sometimes
  - Example: transaction monitor across multiple databases
- Also known as *vertical distribution*

Alternate Architectures: Horizontal



- Rather than vertical, distribute servers across nodes
  - Example: Web server "farm" for load balancing
  - Clients, too (peer-to-peer systems)
  - Most effective for read-heavy systems (cache consistency)

Outline

- Overview (done)
- Goals (done)
- Software (done)
- Client Server (done)
- The Cloud (next)

Ying Lu, UNL, *CSCE990 Advanced Distributed Systems Seminar*
http://cse.unl.edu/~ylu/csce990/notes/Introduction.ppt

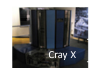
Distributed Computing (1 of 2)

- The Problem
  - Want to run compute/data intensive task
  - But don't have enough resources to run job locally
    - At least, to get results within sensible timeframe
  - Would like to use another, more capable resource
- Solution → Distributed Computing

Images: nasaimages, Extra Ketchup, Google Maps, Dave Page

Distributed Computing (2 of 2)

- Compute *and* data – if you need more, you go somewhere else to get it
- Olden times - Small number of "fast" computers
  - Very expensive
  - Centralized
  - Used nearly all time
  - Time allocations for users



Cray-1 1976 - $8.8 mill, 160 MFLOPS, 8MB RAM
- PS4 ~1 TFLOP
- Smartphones ~200 MFLOPS

- Modern times
  - Cloud and Grid (next)

"Cloud" & "Grid" – Utility Computing?

The Grid…

Is it really like electric grid?

The Cloud…

Is it more like a fog?

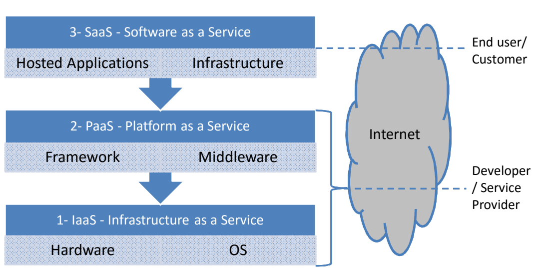
Both about providing access to compute and data resources

What is Cloud Computing?

- Many ways to define it (maybe one for every supplier of "cloud")
- Key characteristics:
  - On demand, dynamic allocation of resources – "elasticity"
  - Abstraction of resource
  - Self-managed
  - Billed for *what you use,* e.g., CPU, time, storage space
  - Standardized interfaces

[FZRL08] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proceedings of Grid Computing Environments Workshop (GCE),* Austin, TX, USA, Nov. 2008, pp. 1–10

Cloud Architecture



- Cloud computing can deliver at any of these levels
- These levels are often blurred and routinely disputed!
- Resources provided on demand

IaaS – Infrastructure as a Service

- User gets access to (usually) virtualised hardware
  - Servers, storage, networking
  - Operating system
- User responsible for managing OS, middleware, runtime, data, application (development)
- e.g., Amazon EC2
  - Get complete virtualized PC (e.g., Linux instance)

Amazon EC2 – The Idea

- EC stands for *Elastic Computing*
- Sign up, then select & configure virtualized resources
  - Machine (OS): Windows Server, OpenSolaris, Fedora, Ubuntu, Debian, SUSE, Gentoo, Amazon Linux AMI
  - Infrastructure:
    - Data: IBM DB2, IBM Informix, Microsoft SQL, MySQL, Oracle
    - Web Hosting: Apache HTTP, IIS/Asp.NET, IBM WebSphere
    - Batch Processing: Hadoop, Condor, Open MPI
  - Newer addition - development environments:
    - IBM sMash, Ruby on Rails, Jboss Enterprise Application Platform
  - Moving towards platform service (PaaS)! (Already there?)
- Additional Web services
  - S3: Simple Storage Solution – transfer data in/out, 1 byte to 5 TB (e.g., DropBox)
  - SQS: Simple Queue Service – transfer between cloud components

Amazon EC2: Pricing

- Free! (at start):
  - Run single Amazon Micro Instance for year
  - 750 hours of EC2, 750 hours of Elastic Load Balancing plus 15 GB data processing
  - 15 GB bandwidth in/out across all services
- On demand instances:
  - Pay per hour, no long-term commitment
  - From $0.025/hour → $0.76/hour
- Reserved instances:
  - Upfront payment, with discount per hour
  - From $227/year + $0.01/hour → $1820/year + $0.32/hour
- Spot instances:
  - Bid for unused EC2 capacity:
  - Spot price fluctuates with supply/demand, if bid over Spot Price, you get it
  - From $0.007/hour → $0.68/hour

EC2 Application Examples

- Peter Harkins (Senior Engineer at The Washington Post)
  - 200 EC2 instances (1,407 server hours)
  - Convert 17,481 pages of Clinton's travel docs within 9 hours after release
- Airbnb
  - 200 EC2 instances
  - 50 BG data daily, S3 for storage (10 TB user pictures)
- Others
  - Zynga, Netflix, Adobe

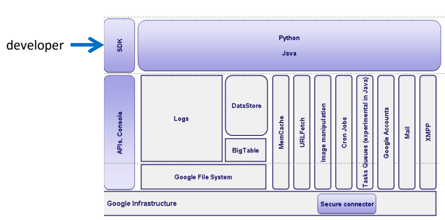
Case studies: http://aws.amazon.com/solutions/case-studies

PaaS – Platform as a Service

- Integrated development environment
  - e.g., application design, testing, deployment, hosting, frameworks for database integration, storage, app versioning, etc.
- Develop applications on top
- Responsible for managing data, application (development)
- Example - Google App Engine

Google App Engine: The Idea

- Sign up via Google Accounts
- Develop App Engine Web applications locally using SDK – emulates all services
- Includes tool to upload application code, static files and config files
- Can 'version' web application instances
- Apps run in Java/Python 'sandbox'
- Automatic scaling and load balancing – abstract across underlying resources



Google App Engine: Pricing

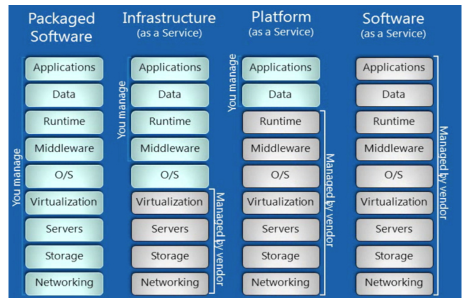
- Free within quota:
  - 500MB storage, 5 million page views a month (~6.5 CPU hours, 1GB)
  - 10 applications/developer
- Billed model:
  - Each app $8/user (max $1000) a month
  - For each app:

Resource	Unit	Unit cost
Outgoing bandwidth	GB	$0.12
Incoming bandwidth	GB	$0.10
CPU Time	CPU hours	$0.10
Stored Data	GB/month	$0.15
High Replication Stg.	GB/month	$0.45
Recipients Emailed	Recipients	$0.0001
Always On	N/A (daily)	$0.30

SaaS – Software as a Service

- Top layer consumed directly by end user – the 'business' functionality
- Application software provided, you configure it (more or less)
- Various levels of maturity:
  - **Level 1:** each customer has own customised version of application in own instance
  - **Level 2:** all instances use same application code, but configured individually
  - **Level 3:** single instance of application across all customers
  - **Level 4:** multiple customers served on load-balanced 'farm' of identical instances
  - Levels 3 & 4: separate customer data! (Somewhat similar to PaaS)
- e.g. Gmail, Google Sites, Google Docs, Facebook

Summary of Provision



Application Migration – adopt level needed

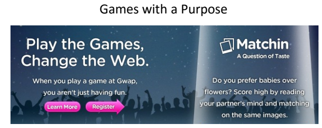
Cloud Open Standards

- Implementations typically have proprietary standards and interfaces
  - Vendors like this – often locked into one implementation
- Community 'push' towards open cloud standards:
  - Open Grid Forum (OGF) – Open Cloud Computing Interface (OCCI)
  - Distributed Management Task Force (DMTF) – Open Virtualisation Format (OVF)

Also HuaaS – Human as a Service

- Extraction of information from crowds of people
- Arbitrary (e.g., notable YouTube videos, digg)
- On-demand task

Games with a Purpose



Amazon Mechanical Turk

Where to Apply Distributed Systems

Application Domain	Associated Networked Application
Finance and commerce	E-commerce (e.g., Amazon and eBay, PayPal), online banking and trading
The information society	Web information and search engines, e-books, Wikipedia; social networking: Facebook and Instagram, Twitter.
Creative industries and entertainment	Online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	Health informatics, on online patient records, monitoring patients
Education	E-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	Sensor technology to monitor earthquakes, floods or tsunamis