Distributed Systems

01. Introduction

Paul Krzyzanowski
Rutgers University
Fall 2014

What can we do now that we could not do before?

1985:
The Internet is 16 years old

Technology advances

Networking

Processors

Memory

Storage

Protocols


June 1976: Robert Metcalfe presents the concept of Ethernet at the National Computer Conference

1980: Ethernet introduced as de facto standard (DEC, Intel, Xerox)

Network architecture

LAN speeds

– Original Ethernet: 2.94 Mbps
– 1985: thick Ethernet: 10 Mbps
  1 Mbps with twisted pair networking
– 1991: 10BaseT - twisted pair: 10 Mbps
  Switched networking: scalable bandwidth
– 1995: 100 Mbps Ethernet
– 1998: 1 Gbps (Gigabit) Ethernet
– 1999: 802.11b (wireless Ethernet) standardized
– 2001: 10 Gbps introduced
– 2005-2010: 40/100 Gbps

348 – >35,000x faster

Network Connectivity

Then:

– Large companies and universities on Internet
– Gateways between other networks
– Consumers had dial-up bulletin boards
– 1985: 1,961 hosts on the Internet

848 million more hosts

Now:

– One Internet (mostly)
– As of July 2014: 1,026,544,414 hosts on the Internet
– Widespread connectivity
– High-speed WAN connectivity: >50-100 Mbps
– Switched LANs
– Wireless networking
Network Connectivity

Metcalfe's Law

The value of a telecommunications network is proportional to the square of the number of connected users of the system.

This makes networking interesting to us!

Computing Power

Computers got...
- Smaller
- Cheaper
- Power efficient
- Faster

Microprocessors became technology leaders

Network Content: Music

Example: 4,207 Billboard hits
- 16 GB
- Average song size: 4.4 MB

Today
- Streaming (Pandora/Spotify): 96-320 kbps
- Download time per song @100 Mbps: ~0.4 seconds
- Storage cost for the collection: ~$0.70 ($154 for a 4 TB drive)

~30 years ago (1985)
- Streaming not practical
- Download time per song, V90 modem @44 Kbps: 15 minutes
- Storage cost: $180,000 (40 MB at $4500)

GPUs scaled as well:
NVIDIA GF100: 3 billion transistors
16 processors each with 32 cores
Network Content: Video

Today
- Netflix streaming 4K video @ 15.6 Mbps (HEVC/h.265 codec)
- YouTube: stores ~76 PB (76×10^15) per year

~30 years ago (1985)
- Video streaming not feasible

Protocols

Faster CPU →
- more time for protocol processing
  - ECC, TCP checksums, parsing
  - Image, audio compression feasible

Faster network →
- bigger (and bloated) protocols
  - e.g., SOAP/XML – human-readable, explicit typing

Flynn’s Taxonomy (1966)

Number of instruction streams and number of data streams

SISO
- traditional uniprocessor system

SIMD
- array (vector) processor
  - Examples:
    - GPUs – Graphical Processing Units for video
    - AVX: Intel’s Advanced Vector Extensions
    - GPGPU (General Purpose GPU); Amdahl, NVIDIA

MISO
- Generally not used and doesn’t make sense
  - Sometimes (rarely!) applied to classifying redundant systems

MIMD
- multiple computers, each with:
  - program-counter, program (instructions), data
- parallel and distributed systems

Subclassifying MIMD

memory
- shared memory systems: multiprocessors
- no shared memory: networks of computers, multicomputers

interconnect
- bus
- switch

delay/bandwidth
- tightly coupled systems
- loosely coupled systems

Parallel Systems: Multiprocessors

- Shared memory
- Shared clock
- All-or-nothing failure
Bus-based multiprocessors

**SMP**: Symmetric Multi-Processing
All CPUs connected to one bus (backplane)

Memory and peripherals are accessed via shared bus. System looks the same from any processor.

The bus becomes a point of congestion ... limits performance

- The **cache**: great idea to deal with bus overload & memory contention
  - Memory that is local to a processor
- CPU performs I/O to cache memory
  - Access main memory only on cache misses

---

Working with a cache

CPU A reads location 12345 from memory

CPU A modifies location 12345

Gets old value

Memory not coherent!

Write-through cache

Fix coherency problem by writing all values through bus to main memory

CPU A modifies location 12345 – write-through

Main memory is now coherent
Write-through cache ... continued

CPU B reads location 12345 from memory
- loads into cache

CPU A
12345: 3

CPU B
12345: 3

Device I/O
12345: 3

Snoopy cache

Add logic to each cache controller:
monitor the bus for writes to memory

Virtually all bus-based architectures use a snoopy cache

Switched multiprocessors

Divide memory into groups and connect chunks of memory to the processors with a crossbar switch

n² crosspoint switches – expensive switching fabric
We still want to cache at each CPU – but we cannot snoop!

Switched multiprocessors

Hierarchical Memory System
- All CPUs see the same address space
- Each CPU has local connectivity to a region of memory – fast access
- Access to other regions of memory – slower
- Placement of code and data becomes challenging – Operating system has to be aware of memory allocation and CPU scheduling

NUMA
NUMA

- SGI Origin’s ccNUMA
- AMD64 Opteron
  - Each CPU gets a bank of DDR memory
  - Inter-processor communications are sent over a HyperTransport link
- Intel
  - Integrated Memory Controller (IMC): fast channel to local memory
  - QuickPath Interconnect: point-to-point interconnect among processors
- Linux 2.5 kernel & onward
  - Multiple run queues
  - Structures for determining layout of memory and processors
- Also supported in Windows Server 2003-2014, Windows 7/8/9, Oracle, SQL Server

NUMA Cache Coherence NUMA: Intel Example

- Home Snoop: Home-based consistency protocol
  - Each CPU is responsible for a region of memory
  - It is the “home agent” for that memory
  - Each home agent maintains a directory (table) that track of who has the latest version

1. CPU sends request to home agent
2. Home agent requests status from the CPU that may have a cached copy (caching agent)
3. (a) Caching agent sends data update to new caching agent
   (b) Caching agent sends status update to home agent
4. Home agent resolves any conflicts & completes transaction

Networks of computers

- Eventually, other bottlenecks occur
  - Network, disk
- We want to scale beyond multiprocessors
  - Multicomputers
- No shared memory, no shared clock
- Communication mechanism needed
  - Traffic much lower than memory access
  - Network

Bus-based multicomputers

Collection of workstations on a LAN

A shared bus-based interconnect gives us the option of snooping
Switched multicomputers

Collection of workstations on a LAN

A switched interconnect does not allow snooping

Wide Area Distribution

Don’t expect to snoop on data traffic

What is a Distributed System?

A collection of independent, autonomous hosts connected through a communication network.

– No shared memory (must use the network)
– No shared clock

Single System Image

Collection of independent computers that appears as a single system to the user(s)

– Independent = autonomous
– Single system: user not aware of distribution

You know you have a distributed system when the crash of a computer you’ve never heard of stops you from getting any work done.

– Leslie Lamport

Why build distributed systems?
How can you get massive performance?

- Multiprocessor systems don’t scale
  - Example: movie rendering
    - Disney’s Cars 2 required 11.5 hours to render each frame (average) – some took 90 hours to render!
    - 12,500 cores on Dell render blades
    - Monsters University required an average of 29 hours per frame
    - Total time: over 100 million CPU hours
    - 3,000 to over 5,000 AMD processors; 10 Gbps and 1 Gbps networks
  - Google
    - Approximately 5.1 billion queries per day
    - Index >50 billion web pages
    - Uses hundreds of thousands of servers to do this

Why build distributed systems?

- Performance ratio
  - Scaling multiprocessors may not be possible or cost effective
- Distributing applications may make sense
  - ATMs, graphics, remote monitoring
- Interactive communication & entertainment
  - Work and play together: email, gaming, telephony, instant messaging
- Remote content
  - Web browsing, music & video downloads, IPTV, file servers
- Mobility
- Increased reliability
- Incremental growth

Design goals: Transparency

High level: hide distribution from users

- Location transparency: users don’t care where resources are
- Migration transparency: resources move at will
- Migration transparency: users cannot tell whether there are copies of resources
- Concurrency transparency: users share resources transparently
- Parallelism transparency: operations take place in parallel without user’s knowledge

Low level: hide distribution from software

- Replication transparency: keeping replicas consistent even if systems go down and come up

Design challenges

Reliability
- Availability: fraction of time system is usable
  - Achieve with redundancy
- Reliability: data must not get lost
  - Includes security

Scalability
- Distributable vs. centralized algorithms
- Can we take advantage of having lots of computers?

Performance
- Network latency, replication, consensus

Programming
- Languages & APIs

Network
- Disconnect, latency, loss of data

Security
- Important but we want convenient access too

Main themes in distributed systems

- Scalability
  - Things are easy on a small scale
    - Geographic latency (multiple data centers), administration, dealing with many thousands of systems
- Latency & asynchronous processes
  - Processes run asynchronously
    - Some messages may take longer to arrive than others
- Availability & fault tolerance
  - Fraction of time that the system is functioning
  - Dead systems, dead processes, dead communication links, lost messages
- Security
  - Authentication, authorization, encryption

Key approaches in distributed systems

- Divide & conquer
  - Break up data sets and have each system work on a small part
  - Merging results is usually efficient
- Replication
  - For high availability, caching, and sharing data
  - Challenge: keep replicas consistent even if systems go down and come up
- Quorum/consensus
  - Enable a group to reach agreement
Centralized model

- No networking
- Traditional time-sharing system
- Single workstation/PC or direct connection of multiple terminals to a computer
- One or several CPUs
- Not easily scalable
- Limiting factor: number of CPUs in system
  - Contention for same resources (memory, network, devices)

Client-Server model

- Clients send requests to servers
- A server is a system that runs a service
- The server is always on and processes requests from clients
- Clients do not communicate with other clients
- Examples
  - FTP, web, email

Layered architectures

- Break functionality into multiple layers
- Each layer handles a specific abstraction
  - Hides implementation details and specifics of hardware, OS, network abstractions, data encoding,...

Tiered architectures

- Tiered (multi-tier) architectures
  - Distributed systems analogy to a layered architecture
- Each tier (layer)
  - Runs as a network service
  - Is accessed by surrounding layers
- The "classic" client-server architecture is a two-tier model
  - Clients: typically responsible for user interaction
  - Servers: responsible for back-end services (data access, printing,...)
Multi-tier example

- Client
  - User interface
  - Data presentation & validation
- Middle tier
  - Queuing requests
  - Coordinating a transaction among multiple servers
  - Managing connections
  - Formatting/converting data
- Back end
  - Database system
  - Legacy software

Peer-to-Peer (P2P) Model

- No reliance on servers
- Machines (peers) communicate with each other
- Goals
  - Robustness
  - Expect that some systems may be down
  - Self-scalability: the system can handle greater workloads as more peers are added
- Examples
  - BitTorrent, Skype

Hybrid model

- Many peer-to-peer architectures still rely on a server
  - Look up, track users
  - Track content
  - Coordinate access
- But traffic-intensive workloads are delegated to peers

Processor pool model

- Collection of CPUs that can be assigned processes on demand
- Render farms
Cloud Computing (Jargon – no precise meaning)

• Provide users with access to:
  – Storage capacity
  – Processing (instances of virtual machines)
  – Network bandwidth

• Services may be geographically distributed systems
  – Ideally, fault tolerant

• Build a “supercomputer” on the fly via networked, loosely coupled computers
  – Scale to needed capacity

Cloud Computing

Resources are provided as a network (Internet) service

  – Software as a Service (SaaS)
    • Salesforce.com, Google Apps, Microsoft Office 365

  – Platform as a Service (Google App Engine, AWS Elastic Beanstalk

  – Storage (Dropbox, Box, Google Drive, OneDrive, …)

The end