CS 555: Distributed Systems
[Distributed Servers & Messaging Systems]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Is each thread assigned the same amount of space?
  - Can this space change dynamically over time?
- Is it possible that space is lost because a thread is not utilizing all the space allocated to it?
  - Internal fragmentation

Topics covered in this lecture

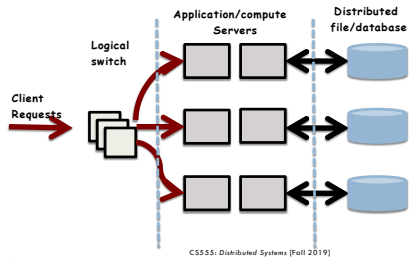
- Distributed Servers
- Performance
- Messaging Systems

Distributed Servers

Mean time for failures and the premise for distributed servers

- Group several machines together
- Don't rely on the availability of any single machine
- Together achieve better stability than each component individually
  - The sum is greater than the parts

Server Clusters



Server Clusters

- Switch is also responsible for **load balancing** requests
  - Simplest way to do this is using round-robin
- If there are different services offered within the cluster?
  - Switch needs to dispatch requests appropriately

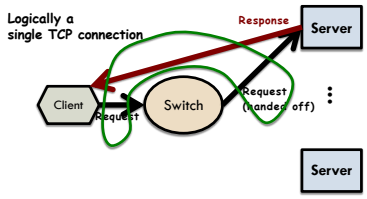
But what about transparency?

- An important consideration is that the server cluster is **transparent**
- Clients typically set up network connections over which requests are sent

But TCP expects an answer from the switch not some arbitrary node

- When server responds to client
  - Inserts **switch's IP address** in source field of the IP packet
- Requires **OS-level modifications**
- Also used in content-aware request distribution

The principle of TCP handoffs



When a cluster offers a single point …

- When there is a failure at that access point?
  - The entire cluster becomes unavailable
- Several access points are typically provided
  - DNS can return **several addresses** all mapped to the same host name
  - Client makes several attempts if there are failures
  - Still requires static access points

Pulls and trade-offs

- Stability
  - Long lived access point
- Flexibility
  - Ability to configure a server cluster including the switch

What would be really nice

- Distributed server with a **dynamically changing** set of machines
- And also varying access points

Mobility support in IP version 6 (MIPv6)

- A **mobile node** has a **home-network**
- This node has a **home-address**
- The node has a **home agent**
  - Takes care of traffic to the mobile node while it is away

Mobility support in IP version 6 (MIPv6)

- When a mobile node attaches to a **foreign network**
  - Gets a temporary **care-of address**
- Care-of address reported to the home-agent
  - **Forward** all traffic to the mobile node

Apps communicating with mobile node only see the home address and not the care-of-address

- Offers a stable address for a distributed server
  - A single, unique contact address is initially assigned
- Contact address is server's **lifetime** address

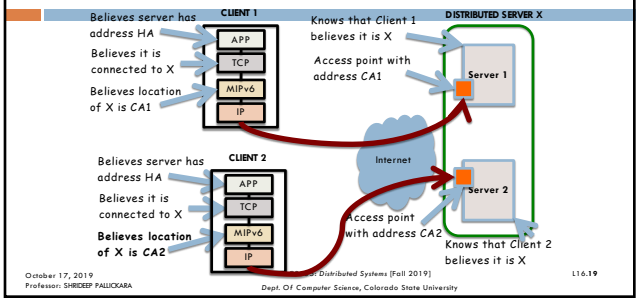
Any node can act operate as the access point

- Record own address as the care-of address
- All traffic will be directed to the access point
- If there's a failure at the access point?
  - Another node takes over
- Potential bottlenecks?
  - Home agent and access point
  - All traffic **must flow** through them

The route optimization feature in MIPv6

- When a mobile node reports its care-of address (CA) to the home-agent (HA)
  - The HA reports the CA to a client
- Client keeps **{HA, CA}**
- Communications will be with the CA
  - Applications can still use the HA
  - MIPv6 protocol stack will translate HA to CA

Depicting Route Optimizations



PERFORMANCE

Measures of performance

- Service time
- Latency
- Throughput
- Capacity
- Efficiency
- Scalability

How fast?

How much?

Performance and Scalability

- Tuning for performance
  - Do **same** work with **less** effort
  - Caching, choice of algorithms $O(n^2)$ to $O(nlogn)$
- Scalability
  - Find ways to parallelize problem
  - Do **more** work with **more** resources

How fast?

How much?

HOW FAST and HOW MUCH

- Separate and can (at times) be at **odds** with each other
- To scale or for better hardware utilization
  - We often end up **increasing** the amount of work for each task
  - Divide tasks into multiple **pipelined** tasks
    - Orchestration overhead

The quest for performance

- What do you mean by **faster**?
- Under what **conditions**?
  - Small or large datasets
  - Perform measurements to substantiate arguments
- **How often** do these conditions arise?
- What are the **hidden costs**?
  - Development/maintenance risks
  - Tradeoffs
  - Ripple effects of decision

Avoid premature optimizations

- First make it **right, then fast**
- **Measure**, don't guess
- Quest for performance is one of the biggest source of **bugs**

AMDAHL'S LAW

How much can we speed things up

- Harvesting crops
  - The more the number of workers
  - The faster the crop can be harvested
- But some things are fundamentally serial
  - Adding additional workers does not make the crop grow faster

The right tool for the right job: Everything is not a nail

- Make sure that problem is **amenable** to parallel decomposition
- Most programs have a **mix** of parallelizable and serial portions

Amdahl's law describes how much a program can be theoretically sped up

- F : Fraction of components that must be executed serially
- N : Number of available processors

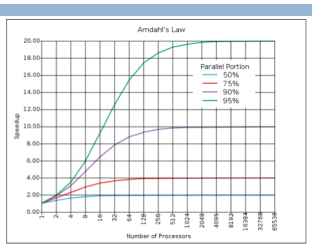
$$Speedup \leq \frac{1}{F + \frac{(1-F)}{N}}$$

$$Utilization = \frac{Speedup}{N}$$

As N approaches infinity; maximum speedup converges to 1/F

- With 50% serial code
  - Maximum speedup is 2
- With 10% serial code
  - Maximum speedup is 10
  - With N= 10
    - Speedup = 5.3 at 53% utilization
  - With N= 100
    - Speedup = 9.2 at 9% utilization

Speedups for different parallelization portions



Source: http://en.wikipedia.org/wiki/Amdahl's_law

Know what to speed up

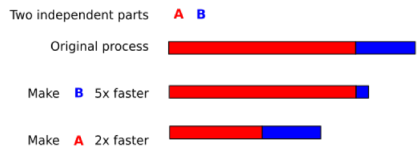


Image from: http://en.wikipedia.org/wiki/Amdahl's_law

MESSAGING SYSTEMS

Several distributed systems built on top of the service offered by transport layer

- Sockets
- Message Passing Interface (MPI)

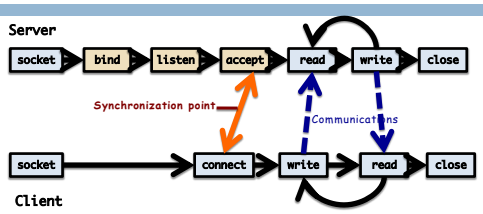
Interfaces to the transport layer

- **Sockets** interface
  - Introduced in the 1970s in Berkley Unix
- X/Open Transport Interface (`XTI`) by AT&T
- Sockets and `XTI` are very similar

Socket primitives for TCP/IP

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

The communication pattern using sockets



Message Passing Interface (MPI)

- High performance computers need highly efficient communications
- Primitives must be:
  - At the right level of **abstraction**
  - Implementation must have **minimal overhead**

Sockets were deemed inefficient …

- **Wrong level** of abstraction
  - Only simple {`send`, `receive`} primitives
- Performance issues
  - Designed for network communications
  - Uses *general-purpose* stacks
  - Not suitable for proprietary protocols in HPC

Result?

- Most HPC systems were shipped with *proprietary* communication libraries
  - High-level and efficient communication primitives
- Of course they were all **mutually incompatible**
  - *Portability* became an issue

Need for hardware and platform independence led to a standard

- **Message Passing Interface** (MPI)
- Designed for parallel applications
- Tailored for transient communications
- Assumes process crashes/partitions are **fatal**
  - No automatic recovery

MPI assumes communications takes place within a group of processes

- Each group is assigned an identifier
- Each process within a group has an identifier
- (`groupId`, `processId`) pair
  - Uniquely indentifies endpoints
  - **Instead of** transport-level addresses

Some of the message passing primitives in MPI

Primitive	Meaning
`MPI_bsend`	Append outgoing message to local send buffer
`MPI_send`	Send message and wait until it is copied to local or remote buffer
`MPI_ssend`	Send message and wait until receipt starts
`MPI_sendrecv`	Send message and wait for reply
`MPI_isend`	Pass reference to outgoing message, and continue
`MPI_issend`	Pass reference to outgoing message Wait until receipt starts
`MPI_recv`	Receive a message; block if there is none
`MPI_irecv`	Check if there is an incoming message; But do not block

MPI offers a very large number of communication primitives

- Official reason
  - More possibilities to improve performance
  - **Pick and choose** most suitable one
- Cynical view
  - Committee could not make up its collective mind
  - Threw in everything!

MULTICAST COMMUNICATIONS

Multicast communications

- Uses datagram packets
- Sender sends packet to multicast address
  - **`224.0.0.0`** to **`239.255.255.255`** **Class D**
- IPv6
  - Multicast addresses have the prefix ff00::/8

Bits	8	4	4	112
Field	Prefix	Flags	Scope	Group ID

Multicast communications: Routing data

- Routers make sure packet delivered to all hosts in multicast group
  - Choose points where streams are *duplicated*
- Pay attention to **TTL** for the datagrams
  - Maximum number of routers a datagram is allowed to cross

Multicast issues

- **Packet size** restrictions in Multicast
- Practical considerations
  - Turned **off** at several institutions to curb *free-riding*

Multicast is not particularly suitable in some situations

- Consumption patterns change dynamically
  - Groups cannot be pre-allocated
- Representing *consumption profiles* as groups
  - Enormous number of groups – potentially 2^N for N consumers
  - Eliminating impossible groups would still require millions of groups

The contents of this slide-set are based on the following references

- *Java Concurrency in Practice. Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. Addison-Wesley Professional. ISBN: 0321349601/978-0321349606.* [Chapter 11]
- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 6, 10]
- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273.* [Chapter 4, 5]