Frequently asked questions from the previous class survey

- Coping with faulty clocks: Why is N > 3f?
- Are wireless networks ever unable to receive time coordination messages in time?
- Lamport's clock
  - Is system wide?
  - Performance impact
  - Set of processes (e.g. banking) use a single-dimensional clock?
- Is NTP better than other algorithms?
- R&S: How are offsets used?

August 31, 2017

Topics covered in this lecture

- Vector Clocks
- Peer to Peer (P2P) Systems
  - Characteristics
  - Generations
  - P2P middleware and requirements

August 31, 2017

Lamport's Clocks order events based on the happened-before relationship

- If \( a \) happened before \( b \), then \( C(a) < C(b) \)
- But nothing can be said about two events \( a \) and \( b \) by merely comparing their values
- \( C(a) < C(b) \)?
  - Does not mean \( a \) happened before \( b \)

August 31, 2017

Let's look a little closer

- \( T_{snd}(m_i) \): Time \( m_i \) was sent
- \( T_{rcv}(m_i) \): Time \( m_i \) was received
- \( T_{snd}(m_i) < T_{rcv}(m_j) \)
- BUT
- \( T_{snd}(m_i) < T_{rcv}(m_j) \) ?
  - NO

August 31, 2017
Concurrent message transmissions

Sending m3 MAY HAVE depended on m1

But sending of m2 has nothing to do with receipt of m1

Lamport clocks do not capture causality

Vector clocks

- Developed by Mattern [1989] and Fidge [1991] to overcome shortcomings of Lamport's clocks
- A vector clock for a system of N processes is an array of N integers
- Each process keeps its own vector clock VC_i
- Process uses its vector clock to timestamp messages

Causality can be captured by Vector clocks

- Event a is known to causally precede event b iff VC(a) < VC(b)
- Each process P_i maintains a vector VC_i
- VC_i[j] is number of events so far at P_i
- if VC_i[j] = k
  - P_i knows k events occurred at P_j
  - P_i's knowledge of local time at P_j

Causal precedence can be captured by Vector clocks

- Event a is known to causally precede event b iff VC(a) < VC(b)
- VC(a) < VC(b) iff VC(a)[k] ≤ VC(b)[k] for all k and at least one of those relationships is strictly smaller
- Each process P_i maintains a vector VC_i
- VC_i[j] is number of events so far at P_i
- if VC_i[j] = k
  - P_i knows k events occurred at P_j
  - P_i's knowledge of local time at P_j

Vectors are piggybacked along with any messages that are sent

1. Before executing an event (sending, delivering, or internal) P_i executes
   - VC_i[j] = VC_i[j] + 1
2. When P_i sends a message m to P_j
   - Set m's timestamp ts(m) to VC_i after doing (1)
3. After receiving m, process P_j adjusts its vector
   - VC_i[k] = max(VC_i[k], ts(m)[k]) for each k
   - Execute step (1) and deliver

Vector clocks example 1
Vector clocks allow us to determine causality and concurrency.

- Event $a$ happened before event $b$ iff $ts(a) \leq ts(b)$ for each process $i$.
- One of those relationships is strictly smaller.
- If this is not true, events $a$ and $b$ are concurrent.

Vector clocks: Other aspects

- If event $a$ has timestamp, $ts(a)$:
  - $ts(a) - 1$ denotes number of events at $P_i$ that precede $a$.
- When $P_j$ receives message $m$ from $P_i$ with timestamp $ts(m) = VC_i$:
  - $P_j$ knows about number of events at $P_i$ that causally preceded $m$.
  - Also, $P_j$ knows about how many events at other processes have preceded the sending of $m$, and on which $m$ may causally depend.

Vector clocks: Disadvantages

- Storage and message payload is proportional to $N$, the number of processes.
- It’s been shown ([Charron-Bost 1991]) that if we are to tell if two events are concurrent by inspecting timestamps:
  - The dimension of $N$ is unavoidable.

Contrasting totally-ordered and causally-ordered multicasting

- Causally-ordered multicasting is weaker than totally-ordered multicasting.
- If two messages are not in any way related to each other:
  - We do not care about the order in which they are delivered to applications.
  - Could be delivered in different order at different applications.
Using Vector Clocks for causally-ordered multicasting
- Clocks are only adjusted when sending and receiving messages
- Upon sending a message, process \( P_i \) will only increment \( VC[i] \) by 1
- When \( P_i \) delivers a message \( m \) with timestamp \( ts(m) \) it adjusts \( VC[k] \)
  - To \( \max(VC[k], ts(m)[k]) \) for each \( k \)

When process \( P_j \) receives a message \( m \) from \( P_i \)
- Delivery of the message \( m \) to the application layer is delayed until 2 conditions are met:
  1. \( ts(m)[i] = VC[j][i] + 1 \)
     - This means \( m \) is the next message that \( P_j \) was expecting from \( P_i \)
  2. \( ts(m)[k] \leq VC[j][k] \) for all \( k \neq i \)
     - This means that \( P_j \) has seen all messages that have been seen by \( P_i \) when it receives \( m \)

An example showing enforcement of causal communications
- Process A
- \( VC_A = (1, 0, 0) \)
- Process B
- \( VC_B = (1, 1, 0) \)
- Process C
- \( VC_C = (1, 0, 0) \), \( VC_C' = (1, 1, 0) \)

Matrix clocks
- Generalizes the notion of vector clocks
- Processes keep estimates of other processes’ vector time [Raynal & Singhal, 1996]
- Essentially, a vector of vector clocks for each of the communicating processes

P2P systems
- Supports the construction of distributed systems
- Data and computational resources are contributed by many hosts
  - All participate in the provisioning of a uniform service
P2P systems

- Ability to share computing resources, storage, and data
  - Present in computers at the "edges of the internet"
- Have been used in several applications such as
  - File sharing, web caching, information distribution
  - 10s of thousands of machines harnessed by these applications

Goals

- Demand for Internet Services continues to grow
  - Scope for expanding popular services is limited when all hosts must be owned and managed by provider
- P2P systems aim to enable sharing of data and resources at a very large scale
  - They do so by eliminating requirements for separately managed servers and their associated infrastructure

P2P systems provide access to information resources

- Information located on computers throughout a network
- Algorithms for placement and retrieval of objects are a key aspect of system design

Traditional client-server systems

- Single computer or a cluster of tightly-coupled servers
- Simple decisions relating to the placement of resources
  - Scale of service is limited by:
    - Server hardware capacity
    - Network connectivity

The delivered service must be

- Fully decentralized
- Self-organizing
- Dynamically balance storage and processing loads between all participating computers
  - Even as computers join and leave the service

Characteristics of P2P Systems

- Fully decentralized
- Self-organizing
- Dynamically balance storage and processing loads between all participating computers
  - Even as computers join and leave the service
P2P characteristics
- Each node contributes resources to the system
- Each node may differ in the quality of the resource that they contribute
- But every node has the same functional capabilities and responsibilities
- Correct operation does not depend on the existence of any centrally administered systems
- Can be designed to provide a limited degree of anonymity to providers and users of resources

Key issue for the efficient operation of P2P systems
- Choice of algorithm for the placement of data across many hosts
- Subsequent access to the data in a manner that balances workload
  - Ensure availability without adding undue overheads

Coping with volatile resources in P2P systems
- Computers and network connections in P2P systems are owned by different entities
  - A single node can become unavailable at any time
- P2P systems do not rely on guaranteed access to individual resources
- They are designed to make probability of failure to access a copy of a replicated object arbitrarily small
  - Degree of resistance to tampering by malicious nodes

Realizing the potential of P2P systems
- Emerged when significant number of users had acquired always-on, broadband connections
  - Made their desktops suitable for resource sharing
- Timelines
  - In the US, this occurred around 1999
  - By mid-2004, worldwide number of broadband connections exceeded 100 million

P2P Generations
- 1st Generation
  - Napster music exchange service
- 2nd Generation
  - Offered greater scalability, anonymity, and fault tolerance
    - Freenet, Gnutella, and BitTorrent
The 3rd Generation of P2P systems

- Emergence of middleware layers for application independent management of distributed resources
- Examples
  - Chord [Stoica et al. 2001]
  - Pastry [Rowstron and Druschel 2001]
  - Tapestry [Zhao et al. 2004]
  - Kademlia [Maymounkov and Mazieres 2002]

Unlike 2nd generation systems, 3rd generation P2P systems

- Provide guarantees of delivery for requests in a bounded number of network hops
- Place replicas of resources on hosts in a structured manner taking account of their:
  - Volatility availability
  - Variable trust worthiness
  - Requirements for load balancing
  - Locality of information storage and use

Use of objects with changing values

- Is much more challenging
- Requires addition of trusted servers to manage sequence of versions
  - Use this to identify the most current version

3rd Generation P2P systems: Resources are identified by globally unique identifiers (GUIDs)

- Derived as a secure hash from some or all of the resource's state
- Make a resource self-certifying
  - Clients receiving a resource can check the validity of the hash
  - Protects it against tampering by untrusted nodes on which it might be stored
  - Requires that states of the resource are immutable
  - Change to the state will result in a different hash value

Availability

- Must avoid situations in which all replicas of an object are simultaneously unavailable
- Use of randomly generated GUIDs assists by distributing object replicas
  - To randomly located nodes
  - If the underlying network spans multiple domains?
  - Risk of simultaneous unavailability is reduced significantly
P2P middleware is designed to orchestrate
- Automatic placement of resources (data items, objects, files, etc.)
- Subsequent location (discovery) of distributed resources

How different P2P generations cope with this issue
- 1st Generation
  - Maintain a centralized index of available files
  - Files are stored at the peers
- 2nd Generation
  - Systems such as Gnutella & Freenet employ partitioned distributed indexes
- 3rd Generation
  - Rely on Overlays

Requirements for P2P systems
- Functional
  - Specific behaviors or functions that must be supported
- Non-functional (or evaluation metrics)
  - Criteria that can be used to judge the operation of a system

Functional requirements for P2P middleware
- Locate and communicate with any resource made available to the system
- Even though resources are dispersed over a large number of nodes
- The ability to add and remove both resources and nodes at will

Non-functional requirements for P2P systems
- Scalability
- Load balancing
- Dynamic host availability
Non-functional requirements:

Load balancing

- Achieved via random placement of resources
- Replicas of heavily used resources are created

Accommodate highly dynamic host availability

- Host computers are free to join or leave at any time
- Provide a dependable service, from unreliable nodes
- As nodes join the system
  - Must be integrated into the system
  - Load must be redistributed to exploit their capabilities
- As nodes leave the system (voluntarily or involuntarily)
  - Redistribute their load and resources
  - Replication levels for some resources must be preserved

Systems that we will observe closely

- 1st Generation
  - Napster
- 3rd Generation
  - Chord
  - Pastry
  - Tapestry
- Unstructured P2P or 2nd Generation
  - Gnutella and BitTorrent

Napster

- First application in which demand for massively scalable storage and retrieval arose
- Downloading of digital music files
- Became very popular soon after its launch
- At its peak
  - Several million users
  - Thousands swapped music files simultaneously
Key features of the architecture

- Centralized indexes
- Users supplied the files
  - Stored and accessed on their personal computers
- Clients add their own music files to the pool of shared resources
  - Transmit a link to Napster’s indexing service for each available file
  - Shared resources at the “edge of the internet”

Napster Architecture

1. File location request
2. List of peers offering the file
3. File Request
4. File Delivered
5. Index Update

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