CS 555: DISTRIBUTED SYSTEMS
[Messaging Systems]

Shrideep Pallickara
Computer Science
Colorado State University

Topics covered in this lecture

- Messaging Systems
  - Publish-subscribe systems
  - Queuing Systems
  - Streams
  - Gossip/Epidemic protocols

Publish subscribe systems

- This is the most widely used of all indirect communications
- Many systems may map to:
  - Request/response or the remote invocation pattern of interactions
  - However, a lot of systems don’t map well into the above models
- Such systems are naturally modeled by the more decoupled style offered by events

Roles in the publish/subscribe system

- Publishers publish structured events to an event service
- Subscribers express interest in particular events through subscriptions
- Subscriptions can be arbitrary patterns over structured events
The key task of a publish/subscribe system

- Match subscriptions against published events
- Ensure delivery of event notifications to subscribers with matching subscriptions

Applications of publish/subscribe systems

- Financial information systems
- Live feeds of real-time data
- Ubiquitous computing
- Monitoring applications
  - Involving sensors and such

Characteristics of publish/subscribe systems

- Heterogeneity
  - Interoperation in a heterogeneous system becomes much simpler
  - Need interfaces for resolving and dealing with events
- Asynchronicity
  - Publishers do not have to synchronize with subscribers while sending events
  - Publishers and subscribers are decoupled

Programming model in publish/subscribe systems

- Event e
  - Publishers disseminate events through a publish(e) operation
  - Subscribers express interest in a set of events through subscriptions:
    - subscribe(f)
    - f refers to a filter that defines a pattern over the set of all possible events
    - Subscriptions can be revoked using unsubscribe (f)

Types of publish/subscribe systems

- Channel-based
  - Publishers publish to named channels
  - Subscribers subscribe to one of the named channels
  - Receive all events on that channel
- Topic-based
  - Notification is expressed in terms of a number of fields
  - One of these fields denotes the topic
Difference between channel and topic-based approaches

- Topics are implicitly defined in channel-based approaches
  - Channels generally implemented as distributed objects
- Topics are explicitly defined in the case of topic-based approaches
  - Topic-based approaches also enhance this using hierarchical organization of topics
  - E.g. /Sports, /Sports/NBA, /Sports/NBA/Denver

Types of publish/subscribe systems

- Content-based
  - Allow expression of subscriptions over a range of fields in the event
  - Filter here is a set of constraints over the values of event attributes
- Type-based
  - Objects must be of a specific type
  - Matching is based either on the type or subtypes
  - Tends to be integrated tightly with programming languages

Centralized vs. Distributed Publish/subscribe substrates

- Centralized publish/subscribe substrates
  - Single server
  - Publishers publish events to central server
  - Subscriber sends subscriptions to that server
  - The single server then performs the matching of subscriptions with the published events
  - Disadvantages:
    - Design lacks resilience and scalability
    - Central server introduces performance bottleneck

- Distributed publish/subscribe substrates
  - Centralized broker is replaced by a network of brokers
  - These brokers cooperate in:
    - The routing of events
    - Dissemination of subscriptions
  - Advantages:
    - Survive multiple node failures
    - Shown to operate well in internet-scale deployments

Publish/Subscribe systems are often used for scalable, real-time disseminations

- Data dissemination managed by the content dissemination network (CDN)
- Comprises a set of software router nodes
  - Logical Overlay
Publish/Subscribe systems: Subscriptions and Matching

- Subscription predicates allow consumers to specify data of interest to them
  - Could be "/" separated Strings, <tag, value> tuples, SQL or Regular Expression queries
- Matching problem is the opposite of databases
  - Evaluate content against all the stored queries

Publish/Subscribe systems: Generation and discovery

- Consumers often need to know what to subscribe to
  - Discovery services used for locating schema
- Data published by the producers
  - Has to be self descriptive with values for content descriptors

Publish/subscribe systems: An example

Architecture of publish-subscribe systems

Flooding: Publishing events

- Event notification is sent to all nodes
- Matching is performed by the subscriber
  - A subscriber may end up discarding most of the events that it receives
Flooding:
Subscriptions

- All subscriptions are sent to all possible publishers
- Matching is performed by the publisher
  - Publisher then sends matched events directly to the subscribers

Filtering [1/2]

- Brokers forward notifications through CDN only where there is a path to a valid subscriber
- Achieved by
  - Propagating subscription information through the network towards potential publishers
  - Store associated state at each broker
- Each broker maintains
  - List of neighboring brokers
  - List of subscriptions directly serviced by that node
  - Routing table indicating pathways to reach nodes

Filtering [2/2]

- Requires implementation of matching at each node in the network
- Matching function
  - Takes: Event notification and list of nodes with associated subscriptions
  - Returns: Set of nodes where the event notification matches the subscriptions

Rendezvous nodes

- View the set of all possible events as an event space
- Partition responsibility for this event space between a set of brokers in the system
- Rendezvous nodes are broker nodes that are responsible for a subset of the event space
- When an event e is published, REN(e) returns one or more rendezvous nodes
  - Responsible for matching e against subscriptions in the system

Publish/Subscribe Systems

- CORBA: Event Service, Notification Service
- Java Message Service
  - IMS clients are vendor agnostic
  - Vendors do not interoperate with each other
- Wire formats
  - Advanced Message Queuing Protocol
- Web Services
  - WS-Eventing, WS-Notification

Message Queuing
Message queuing systems
- Applications communicate by **inserting messages** in queues
- Messages delivered to destination
  - Even if it was down when message was sent
- Each application has its own private queue

Message queuing guarantees:
**Sender**
- Message will be eventually inserted in recipient's queue
- No guarantees about
  - When
  - If the message will be read
- Time-decoupled interaction

The store-and-forward approach
- Producers **place** messages on to a queue
  - When destination is not available
- Queuing system responsible for **resending** messages from queue
  - When destination is available

Basic interface to a message queuing system
<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>get</td>
<td>Block until specified queue is non-empty, and remove the first message</td>
</tr>
<tr>
<td>poll</td>
<td>Check a specified queue for messages, and remove the first. Never blocks.</td>
</tr>
<tr>
<td>notify</td>
<td>Install a handler to be called when a message is put into the specified queue</td>
</tr>
</tbody>
</table>

Queuing systems: Problems
- Typically **point-to-point** communications
- Store-and-forward does not scale particularly well

STREAM ORIENTED COMMUNICATIONS
Our focus so far

- Exchange of independent, autonomous messages
- RPC calls, replies to request, messaging
- Processes may be slow or fast
  - Timing has no impact on correctness

Several places where timing plays a crucial role

- If sound has been sampled at a certain frequency
- It is essential that the sound is played back at the same frequency
- Playing at a different rate?
  - Incorrect version of the original sound

Support for time-dependent information formulated as support for continuous media

- How is the information represented?
- Encoding
  - Text: Unicode or ASCII
  - Images: JPEG/GIF ...
  - Audio: 16-bit PCM samples

Continuous media

- Temporal relationships are fundamental to interpreting what the data means
- Order and spacing must be preserved

Discrete media

- Temporal relationships are not fundamental to interpreting the data
- Asynchronous transmissions
  - Data transmitted one after the other
  - No other timing constraints regarding inter-packet spacing

Transmissions

- Synchronous transmissions
  - Maximum end-to-end delay defined for each packet
  - No problem if samples are propagated faster
- Isochronous transmissions
  - Packets must be transmitted on time
  - Subject to maximum and minimum end-to-end delay
  - Bounded (delay) jitter
Streams and complexity

- Simple streams
  - Single sequence of data

- Complex stream
  - Several related simple streams
  - Substreams are time-dependent
  - Synchronization needed

Computing Jitter

- The Jitter J is computed based on the RTP RFC
  \[ J = J + \left( \left| D(i, i - 1) \right| - J \right)/16 \]

- \( D(i, i - 1) \) is the difference between the delay for the \( i \)th packet and the delay for the \( (i - 1) \)th packet.

- Values from the past and present play a role in the jitter calculation.

Use of buffering to enforce QoS

- Packets depart from source
  1 2 3 4 5 6 7 8

- Packets arrive at buffer
  1 2 3 4 5 6 7 8

- Packets removed from buffer
  Gap in playback
  1 2 3 4 5 6 7 8

Why interleave packet frames?

<table>
<thead>
<tr>
<th>Sent</th>
<th>Last Packet</th>
<th>Delivered</th>
<th>Lost Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>9 10 11 12 13 14 15 16</td>
<td>1 2 3 4 5 6 7 8</td>
<td>9 10 11 12 13 14 15 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sent</th>
<th>Last Packet</th>
<th>Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>9 10 11 12 13 14 15 16</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

Gossip based data dissemination

- Can disease spread be the basis for disseminating information?
  - Epidemic protocols

- Rapidly propagate information among a large collection of nodes
  - Using only local information
  - No central coordinating component
Epidemic protocols

- The objective is to infect nodes with new information as fast as possible
- A node is infected if it holds data that it is willing to spread
- A node that
  - Has not seen the data: susceptible
  - Is not willing or able to spread: removed

A popular propagation model is anti-entropy

- Node P picks node Q at random
- Exchange of updates
  - P only pushes its own updates to Q
  - P only pulls new updates from Q
  - P and Q send updates to each other (push/pull)

Rapidly spreading updates:

Push-based

- Only pushing updates turns out to be a bad choice
- Updates can only be propagated by infected nodes
  - If many nodes are infected?
    - Probability of selecting a susceptible node is small
  - Nodes remain susceptible for a long time

Pull-based

- Spreading updates is triggered by susceptible nodes
- Chances that a susceptible node contacts an infectious one is higher

In practice: Both push/pull are used

- Round
  - Every node has taken the initiative to exchange updates with a random node
  - Number of rounds to propagate a single update
    - \( \log(N) \)

Gossips: Variant of the epidemic scheme

- Also called rumor spreading
  - If P has been updated for data item x
  - Finds Q and tries to update it
  - If Q was already updated?
    - P may lose interest in spreading the update with \( p=1/k \)
**Gossiping: An excellent way to spread news**

- But cannot guarantee that all nodes will actually be updated
- When a large number of nodes participate in epidemics
  - A fraction of users ($S$) can miss updates
  
  \[ S = e^{-(k+1)(1-s)} \]

**Managing Web Content Delivery**

**Akamai**

- Websites redirect users to Akamaized URLs
- IP address associated with client used to select server-farm closest to client.
  - Most popular content served up from caches
  - Benefits of caching and network proximity
- Server farms sync up with managed websites to track content changes.

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