Introduction to Distributed Systems

27 October 2019
Lecture 1

Slide Credits: Maarten van Steen
Topics for Today

• Course Introduction and Syllabus

• Definitions

• Goals

Source: TvS 1.1 - 1.3.1
SE 424: Distributed Systems Goals

By the end of the course, students will be able to:

1. Define a distributed system and give examples of several different distributed systems paradigms.
2. Design and implement application level communication protocols using TCP or UDP.
3. Design and implement a tool which works in a client-server architecture and uses TCP or UDP for communication.
4. Design and implement a tool which works in a peer-to-peer architecture and uses TCP or UDP for communication.
5. Explain fundamental problems in distributed systems relating to synchronization, mutual exclusion, replication, and fault tolerance.
6. Design and implement applications which communicate using application level multicast and epidemic communication mechanisms.
7. Build and test a distributed communication tool which is based on the Apache Kafka Distributed Streaming Platform.
So Far

• Course Introduction and Syllabus

• Definitions

• Goals
A distributed system is a piece of software that ensures that:

*a collection of autonomous computing elements that appears to its users as a single coherent system*

Two aspects: (1) **Autonomous** computing elements, also referred to as nodes, be they hardware devices or software processes and (2) Single coherent system: users or applications perceive a single system → nodes need to collaborate.

Each node is **autonomous**:

- Its own notion of time → there is no global clock 🕒
- Leads to fundamental synchronization and coordination problems.

Collection of nodes and group:

- How to manage group membership? 👥
- How to know you are communicating with an authorized member?
Organization

Overlay network

- Each node in the collection communicates only with other nodes in the system, its neighbors. The set of neighbors may be dynamic, or may even be known only implicitly (i.e., requires a lookup).

Overlay types

- Well-known example of overlay networks: peer-to-peer systems.
  - **Structured**: each node has a well-defined set of neighbors with whom it can communicate (tree, ring).
  - **Unstructured**: each node has references to randomly selected other nodes from the system.
Coherent system

• The collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system takes place.

• Examples:
  1. An end user cannot tell where a computation is taking place
  2. Where data is exactly stored should be irrelevant to an application
  3. Whether data has been replicated or not is completely hidden

Key: Distribution Transparency

Snag: Partial Failures

• It’s inevitable that at any time only a part of the distributed system fails. Hiding partial failures and their recovery is often very difficult and in general impossible to hide.
Middleware: OS of Distributed Systems

What’s inside?
- Commonly used **components and functions** that need not be implemented by applications separately.
What do we want to achieve?

- Supporting sharing of resources
- Distribution transparency
- Openness
- Scalability
Sharing resources

• Canonical examples
  - Cloud-based shared storage and files
  - Peer-to-peer assisted multimedia streaming
  - Shared mail services (think of outsourced mail systems)
  - Shared Web hosting (think of content distribution networks)

Observation:
• “The network is the computer”
  (quote from John Gage, then at Sun Microsystems)
## Distribution Transparency

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how an object is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where an object is located</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that an object may be moved to another location while in use</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that an object may move to another location</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that an object is replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that an object may be shared by several independent users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hides failure and recovery of objects</td>
</tr>
</tbody>
</table>

**Note:** Distribution transparency is a nice goal, but aiming at full distribution transparency may be too much.
Degree of Transparency

Observation: Aiming at full distribution transparency may be too much.

- There are communication latencies that cannot be hidden
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance, exposing distribution of the system
  - Keeping replicas exactly up-to-date with the master takes time
  - Immediately flushing write operations to disk for fault tolerance
Exposing Distribution

- Exposing distribution may be good
  - Making use of location-based services (finding your nearby friends)
  - When dealing with users in different time zones
  - When it makes it easier for a user to understand what’s going on (when e.g., a server does not respond for a long time, report it as failing).

- Conclusion:
  - Distribution transparency is a nice a goal, but achieving it is a different story, and it should often not even be aimed at.
Openness of Distributed Systems

**Open distributed system**

Be able to interact with services from other open systems, irrespective of the underlying environment:

- Systems should conform to well-defined interfaces
- Systems should support portability of applications
- Systems should easily interoperate

**Achieving openness**

At least make the distributed system independent from heterogeneity of the underlying environment:

- Hardware
- Platforms
- Languages
Policy versus Mechanisms

**Implementing openness**: Support for different policies:
- What level of consistency do we require for client-cached data?
- Which operations do we allow downloaded code to perform?
- Which QoS requirements do we adjust in the face of varying bandwidth?
- What level of secrecy do we require for communication?

**Implementing openness**: Ideally, a distributed system provides only mechanisms:
- Allow (dynamic) setting of caching policies
- Support different levels of trust for mobile code
- Provide adjustable QoS parameters per data stream
- Offer different encryption algorithms
Scale in Distributed Systems

Observation
Many developers of modern distributed systems easily use the adjective “scalable” without making clear why their system actually scales.

Scalability
At least three components:
• Number of users and/or processes (size scalability)
• Maximum distance between nodes (geographical scalability)
• Number of administrative domains (administrative scalability)

Observation
Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.
**Geographical scalability**

<table>
<thead>
<tr>
<th>Cannot simply go from LAN to WAN:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Many distributed systems assume synchronous client-server interactions</td>
</tr>
<tr>
<td>• Client sends request and waits for an answer.</td>
</tr>
<tr>
<td>• Latency may easily prohibit this scheme.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WAN links are often inherently unreliable</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Simply moving streaming video from LAN to WAN will fail.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lack of multipoint communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A simple search broadcast cannot be deployed.</td>
</tr>
<tr>
<td>• Solution: Develop separate naming and directory services (having their own scalability problems).</td>
</tr>
</tbody>
</table>
Administrative scalability

Essence: Conflicting policies concerning usage (and thus payment), management, and security

Examples:
- **Computational grids**: share expensive resources between different domains.
- **Shared equipment**: how to control, manage, and use a shared radio telescope constructed as large-scale shared sensor network?

Exception: Several peer-to-peer networks
- File-sharing systems (based, e.g., on BitTorrent)
- Peer-to-peer telephony (Skype)
- Peer-assisted audio streaming (Spotify)

Note: End users collaborate and not administrative entities.
Size Scalability

- Root causes for scalability problems with centralized solutions

  - Computational capacity, limited by the CPUs
  - Storage capacity, including the transfer rate between CPUs and disks
  - Network between the user and the centralized service
Size Scaling Technique 1

Hide communication latencies
Avoid waiting for responses; do something else:

- Make use of asynchronous communication
- Have separate handler for incoming response

Problem: not every application fits this model
Offloading Work

- Facilitate solution by moving computations to client
Distribution

Partition data and computations across multiple machines:

- Move computations to clients (Java applets)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)
Replication/caching

Make copies of data available at different machines:

- Replicated file servers and databases
- Mirrored Web sites
- Web caches (in browsers and proxies)
- File caching (at server and client)
Observation
Applying scaling techniques is easy, except for one thing:
• Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
• Always keeping copies consistent and in a general way requires global synchronization on each modification.
• Global synchronization precludes large-scale solutions.

Observation
If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.
Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions:

• The network is reliable
• The network is secure
• The network is homogeneous
• The topology does not change
• Latency is zero
• Bandwidth is infinite
• Transport cost is zero
• There is one administrator
Conclusion

- Course Introduction and Syllabus
- Definitions
- Goals