Architectural Patterns for Distributed Systems

Idaho State University | Computer Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Topics Covered

- Architectural patterns for distributed systems

ROAR

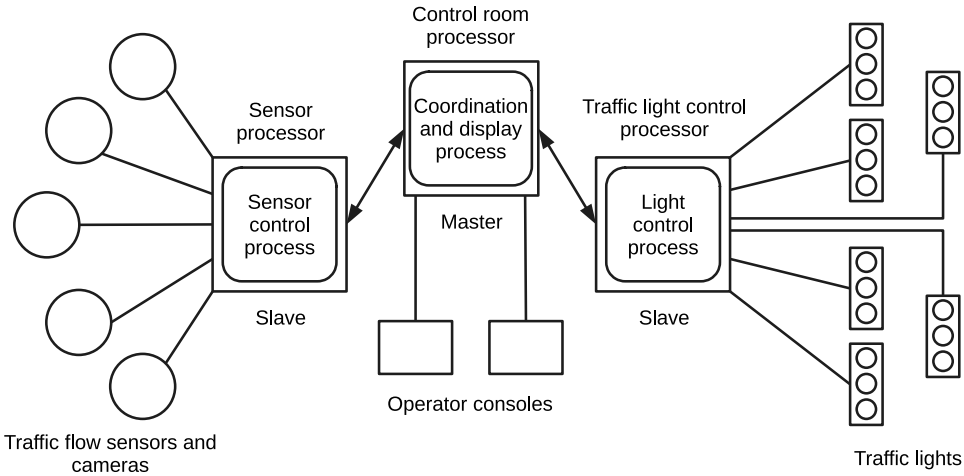
Architectural patterns

- Widely used ways of organizing the architecture of a distributed system:
  - **Master-slave architecture**, which is used in real-time systems in which guaranteed interaction response times are required.
  - **Two-tier client-server architecture**, which is used for simple client-server systems, and where the system is centralized for security reasons.
  - **Multi-tier client-server architecture**, which is used when there is a high volume of transactions to be processed by the server.
  - **Distributed component architecture**, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client-server systems.
  - **Peer-to-peer architecture**, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other

Master-slave architectures

- Master-slave architectures are commonly used in real-time systems where there may be separate processors associated with data acquisition from the system's environment, data processing and computation and actuator management.
- The 'master' process is usually responsible for computation, coordination and communications and it controls the 'slave' processes.
- 'Slave' processes are dedicated to specific actions, such as the acquisition of data from an array of sensors.

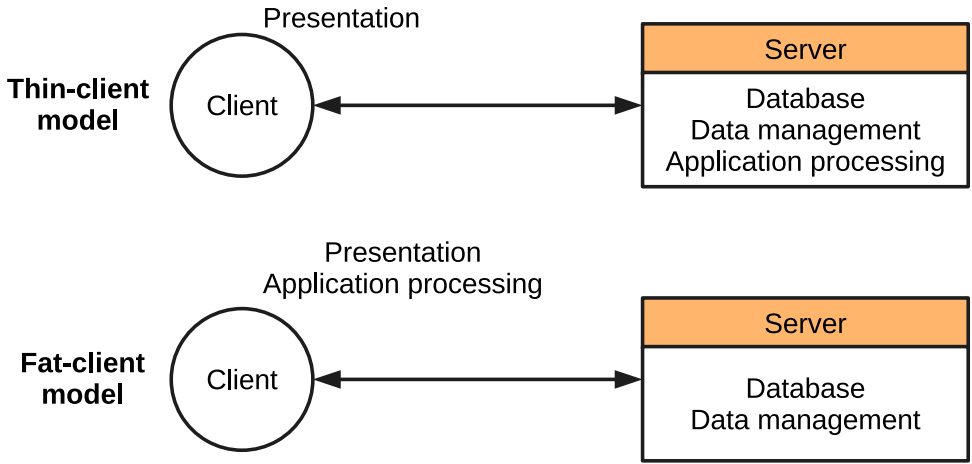
A traffic management system with a master-slave architecture



Two-tier client server architectures

- In a two-tier client-server architecture, the system is implemented as a single logical server plus an indefinite number of clients that use that server.
  - Thin-client model, where the presentation layer is implemented on the client and all other layers (data management, application processing and database) are implemented on a server.
  - Fat-client model, where some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server.

Thin- and fat-client architectural models



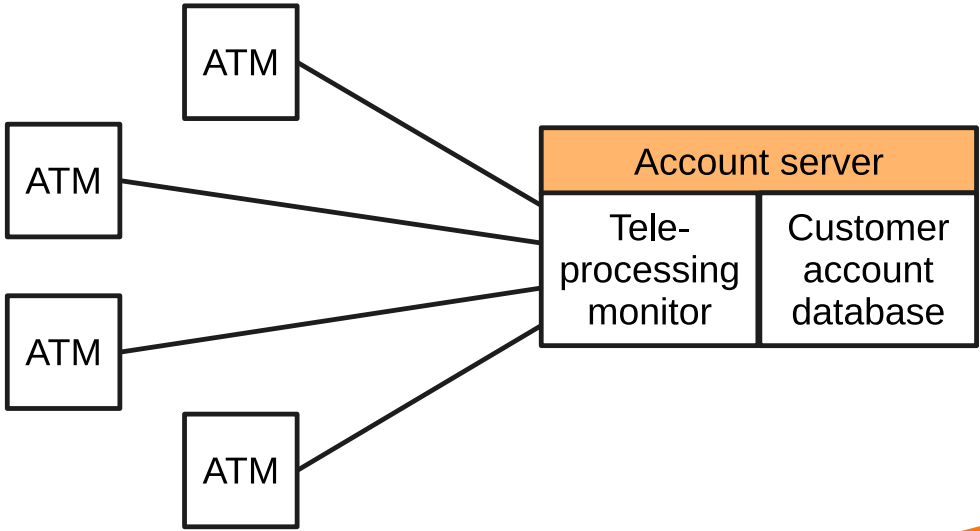
Thin client model

- Used when legacy systems are migrated to client server architectures.
  - The legacy system acts as a server in its own right with a graphical interface implemented on a client.
- A major disadvantage is that it places a heavy processing load on both the server and the network.

Fat client model

- More processing is delegated to the client as the application processing is locally executed.
- Most suitable for new C/S systems where the capabilities of the client system are known in advance.
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients.

A fat-client architecture for an ATM system



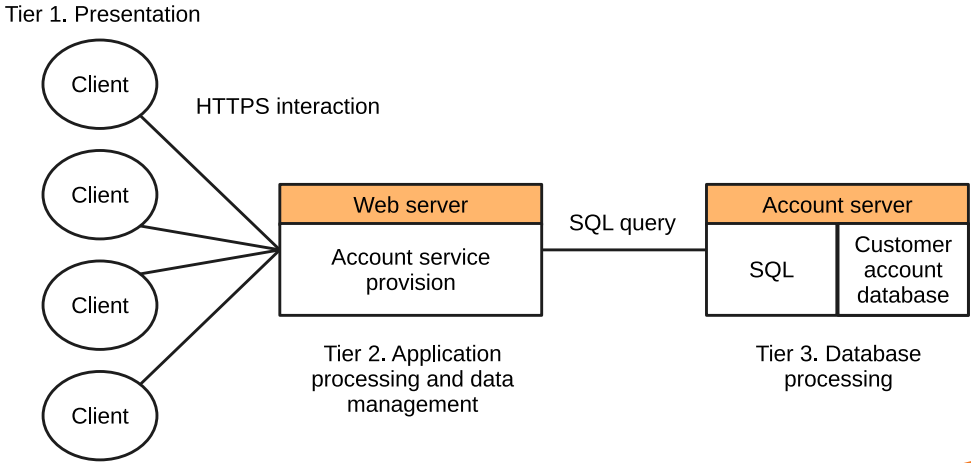
Thin and fat clients

- Distinction between thin and fat client architectures has become blurred
- Javascript allows local processing in a browser so ‘fat-client’ functionality available without software installation
- Mobile apps carry out some local processing to minimize demands on network
- Auto-update of apps reduces management problems
- There are now very few thin-client applications with all processing carried out on remote server.

Multi-tier client-server architectures

- In a ‘multi-tier client–server’ architecture, the different layers of the system, namely presentation, data management, application processing, and database, are separate processes that may execute on different processors.
- This avoids problems with scalability and performance if a thin-client two-tier model is chosen, or problems of system management if a fat-client model is used.

Three-tier architecture for an Internet banking system



Use of client-server architectural patterns

Architecture: Two-tier client–server architecture with thin clients

Applications:

- Legacy system applications that are used when separating application processing and data management is impractical. Clients may access these as services, as discussed in Section 18.4.
- Computationally intensive applications such as compilers with little or no data management.
- Data-intensive applications (browsing and querying) with nonintensive application processing. Browsing the Web is the most common example of a situation where this architecture is used.

Use of client-server architectural patterns

Architecture: Two-tier client-server architecture with fat clients

Applications:

- Applications where application processing is provided by off-the-shelf software (e.g., Microsoft Excel) on the client.
- Applications where computationally intensive processing of data (e.g., data visualization) is required.
- Mobile applications where internet connectivity cannot be guaranteed. Some local processing using cached information from the database is therefore possible.

Architecture: Multi-tier client–server architecture

Applications:

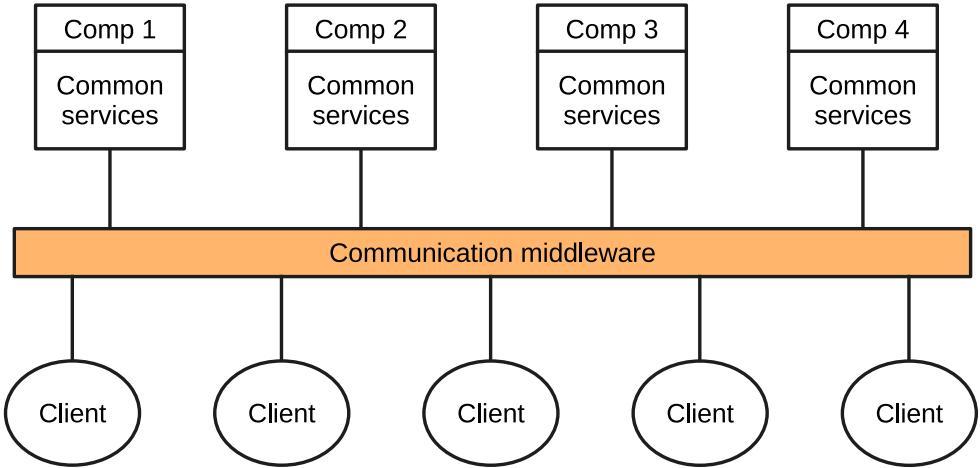
- Large-scale applications with hundreds or thousands of clients

Distributed component architectures

- There is no distinction in a distributed component architecture between clients and servers.
- Each distributable entity is a component that provides services to other components and receives services from other components.
- Component communication is through a middleware system.

Idaho State University Computer Science

A distributed component architecture



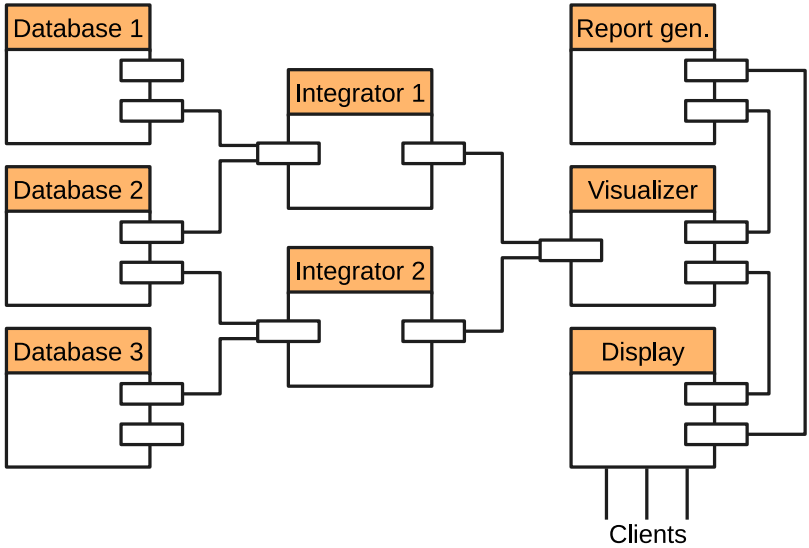
ROAR

Benefits of distributed component architecture

- It allows the system designer to delay decisions on where and how services should be provided.
- It is a very open system architecture that allows new resources to be added as required.
- The system is flexible and scalable.
- It is possible to reconfigure the system dynamically with objects migrating across the network as required.

Idaho State University
Computer Science

A data mining system



ROAR

Disadvantages of distributed component architecture

- Distributed component architectures suffer from two major disadvantages:
  - They are more complex to design than client–server systems. Distributed component architectures are difficult for people to visualize and understand.
  - Standardized middleware for distributed component systems has never been accepted by the community. Different vendors, such as Microsoft and Sun, have developed different, incompatible middleware.
- As a result of these problems, service-oriented architectures are replacing distributed component architectures in many situations.

Peer-to-peer architectures

- Peer to peer (p2p) systems are decentralized systems where computations may be carried out by any node in the network.
- The overall system is designed to take advantage of the computational power and storage of a large number of networked computers.
- Most p2p systems have been personal systems but there is increasing business use of this technology.

Peer-to-peer systems

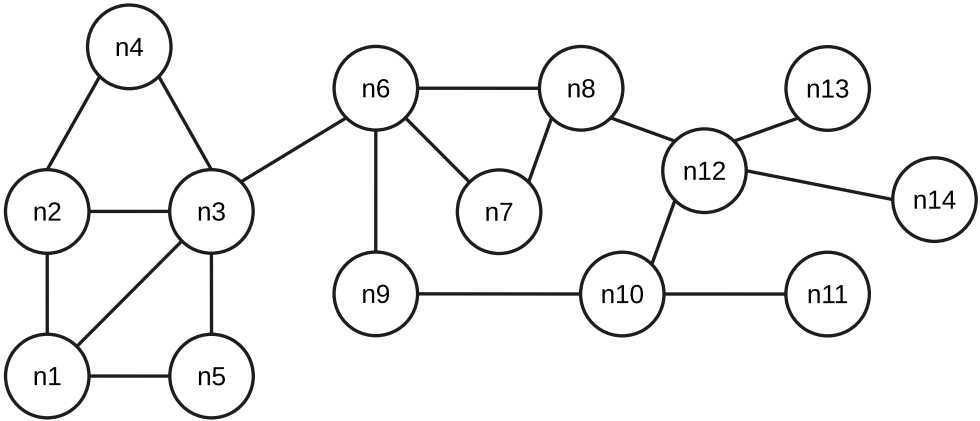
- File sharing systems based on the BitTorrent protocol
- Messaging systems such as Jabber
- Payments systems – Bitcoin
- Databases – Freenet is a decentralized database
- Phone systems – Viber
- Computation systems - SETI@home

P2P architectural models

- The logical network architecture
  - Decentralized architectures;
  - Semi-centralized architectures.
- Application architecture
  - The generic organization of components making up a p2p application.
- Focus here on network architectures.

Idaho State University
Computer Science

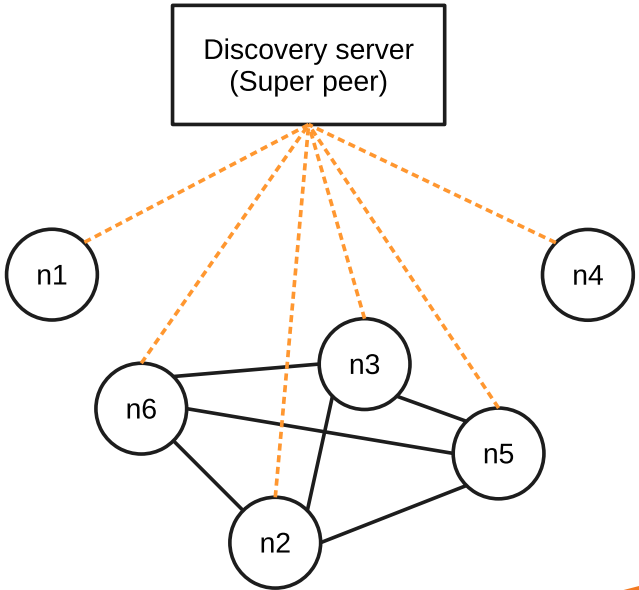
A decentralized p2p architecture



ROAR

Idaho State University
Computer Science

A semi-centralized p2p architecture



ROAR

Key points

- Architectural patterns for distributed systems include master-slave architectures, two-tier and multi-tier client-server architectures, distributed component architectures and peer-to-peer architectures.
- Distributed component systems require middleware to handle component communications and to allow components to be added to and removed from the system.

Are there any questions?