Distributed Software Engineering

Idaho State University | Computer Science

Isaac Griffith

CS 3321
Department of Computer Science
Idaho State University

ROAR

Topics Covered

- Distributed Systems
- Client-server computing

Distributed systems

- Virtually all large computer-based systems are now distributed systems.
  "… a collection of independent computers that appears to the user as a single coherent system."
- Information processing is distributed over several computers rather than confined to a single machine.
- Distributed software engineering is therefore very important for enterprise computing systems.

Distributed system characteristics

- Resource sharing
  - Sharing of hardware and software resources.
- Openness
  - Use of equipment and software from different vendors.
- Concurrency
  - Concurrent processing to enhance performance.
- Scalability
  - Increased throughput by adding new resources.
- Fault tolerance
  - The ability to continue in operation after a fault has occurred.

Distributed systems

Distributed systems issues

- Distributed systems are more complex than systems that run on a single processor.
- Complexity arises because different parts of the system are independently managed as is the network.
- There is no single authority in charge of the system so top-down control is impossible.

Design issues

- **Transparency** To what extent should the distributed system appear to the user as a single system?
- **Openness** Should a system be designed using standard protocols that support interoperability?
- **Scalability** How can the system be constructed so that it is scaleable?
- **Security** How can usable security policies be defined and implemented?
- **Quality of service** How should the quality of service be specified.
- **Failure management** How can system failures be detected, contained and repaired?

Transparency

- Ideally, users should not be aware that a system is distributed and services should be independent of distribution characteristics.
- In practice, this is impossible because parts of the system are independently managed and because of network delays.
  - Often better to make users aware of distribution so that they can cope with problems
- To achieve transparency, resources should be abstracted and addressed logically rather than physically. Middleware maps logical to physical resources.

Openness

- Open distributed systems are systems that are built according to generally accepted standards.
- Components from any supplier can be integrated into the system and can inter-operate with the other system components.
- Openness implies that system components can be independently developed in any programming language and, if these conform to standards, they will work with other components.
- Web service standards for service-oriented architectures were developed to be open standards.

Scalability

- The scalability of a system reflects its ability to deliver a high quality service as demands on the system increase
  - Size It should be possible to add more resources to a system to cope with increasing numbers of users.
  - Distribution It should be possible to geographically disperse the components of a system without degrading its performance.
  - Manageability It should be possible to manage a system as it increases in size, even if parts of the system are located in independent organizations.
- There is a distinction between scaling-up and scaling-out. Scaling up is more powerful system; scaling out is more system instances.

Security

- When a system is distributed, the number of ways that the system may be attacked is significantly increased, compared to centralized systems.
- If a part of the system is successfully attacked then the attacker may be able to use this as a 'back door' into other parts of the system.
- Difficulties in a distributed system arise because different organizations may own parts of the system. These organizations may have mutually incompatible security policies and security mechanisms.

Types of attack

- The types of attack that a distributed system must defend itself against are:
  - Interception, where communications between parts of the system are intercepted by an attacker so that there is a loss of confidentiality.
  - Interruption, where system services are attacked and cannot be delivered as expected.
  - Denial of service attacks involve bombarding a node with illegitimate service requests so that it cannot deal with valid requests.
  - Modification, where data or services in the system are changed by an attacker.
  - Fabrication, where an attacker generates information that should not exist and then uses this to gain some privileges.

Quality of service

- The quality of service (QoS) offered by a distributed system reflects the system's ability to deliver its services dependably and with a response time and throughput that is acceptable to its users.
- Quality of service is particularly critical when the system is dealing with time-critical data such as sound or video streams.
  - In these circumstances, if the quality of service falls below a threshold value then the sound or video may become so degraded that it is impossible to understand.

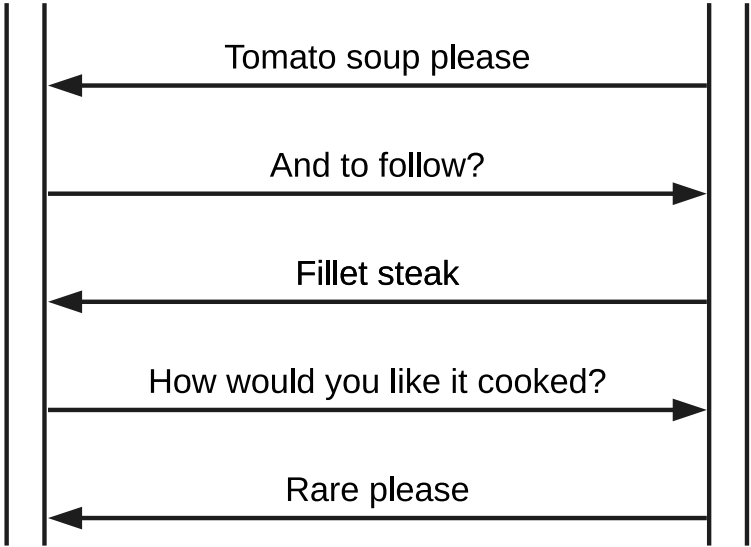
Failure management

- In a distributed system, it is inevitable that failures will occur, so the system has to be designed to be resilient to these failures.
  “You know that you have a distributed system when the crash of a system that you’ve never heard of stops you getting any work done.”
- Distributed systems should include mechanisms for discovering if a component of the system has failed, should continue to deliver as many services as possible in spite of that failure and, as far as possible, automatically recover from the failure.

Models of interaction

- Two types of interaction between components in a distributed system
  - Procedural interaction, where one computer calls on a known service offered by another computer and waits for a response.
  - Message-based interaction, involves the sending computer sending information about what is required to another computer. There is no necessity to wait for a response.

Diner and a waiter



Idaho State University
Computer Science

Waiter and the kitchen messages

```
<starter>
    <dish name="soup" type="tomato" />
    <dish name="soup" type="fish" />
    <dish name="pigeon salad" />
</starter>
<main>
    <dish name="steak" type="sirloin" cooking="medium" />
    <dish name="steak" type="fillet" cooking="rare" />
    <dish name="sea bass">
</main>
<accompaniment>
    <dish name="french fries" portions="2" />
    <dish name="salad" portions="1" />
</accompaniment>
```


ROAR

Remote procedure calls

- Procedural communication in a distributed system is implemented using remote procedure calls (RPC).
- In a remote procedure call, one component calls another component as if it was a local procedure or method. The middleware in the system intercepts this call and passes it to a remote component.
- This carries out the required computation and, via the middleware, returns the result to the calling component.
- A problem with RPCs is that the caller and the callee need to be available at the time of the communication, and they must know how to refer to each other.

Message passing

- Message-based interaction normally involves one component creating a message that details the services required from another component.
- Through the system middleware, this is sent to the receiving component.
- The receiver parses the message, carries out the computations and creates a message for the sending component with the required results.
- In a message-based approach, it is not necessary for the sender and receiver of the message to be aware of each other. They simple communicate with the middleware.

Middleware

- The components in a distributed system may be implemented in different programming languages and may execute on completely different types of processor. Models of data, information representation and protocols for communication may all be different.
- Middleware is software that can manage these diverse parts, and ensure that they can communicate and exchange data.

Middleware support

- Interaction support, where the middleware coordinates interactions between different components in the system
  - The middleware provides location transparency in that it isn't necessary for components to know the physical locations of other components.
- The provision of common services, where the middleware provides reusable implementations of services that may be required by several components in the distributed system.
  - By using these common services, components can easily inter-operate and provide user services in a consistent way.

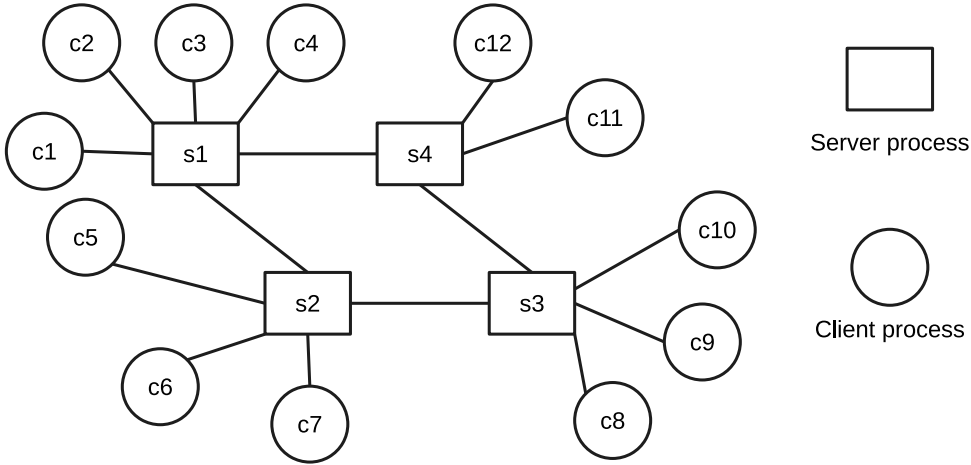
Client-server computing

Client-server computing

- Distributed systems that are accessed over the Internet are normally organized as client-server systems.
- In a client-server system, the user interacts with a program running on their local computer (e.g. a web browser or mobile application). This interacts with another program running on a remote computer (e.g. a web server).
- The remote computer provides services, such as access to web pages, which are available to external clients.

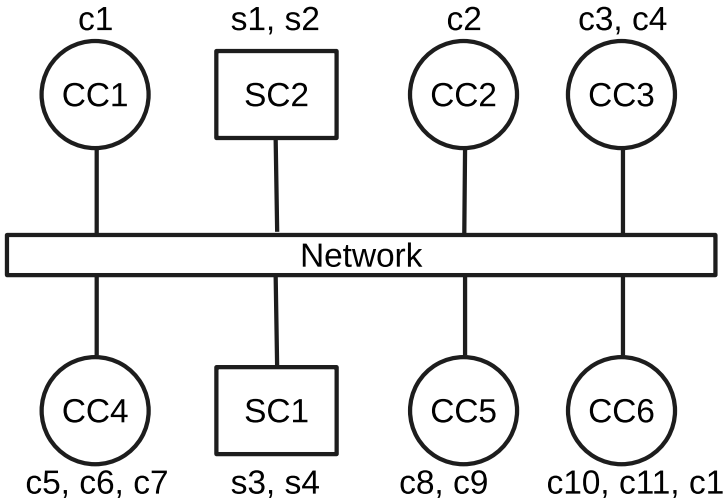
Idaho State University
Computer Science

Client-server interaction



ROAR

Mapping of client and servers to networked computers

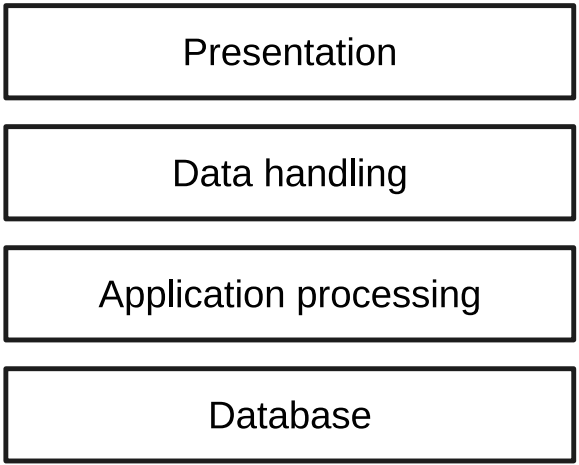


Server computer

Client computer

Idaho State University
Computer Science

Layered architectural model for client-server applications



ROAR

Idaho State University
Computer Science

Layers in a client/server system

- Presentation
  - concerned with presenting information to the user and managing all user interaction.
- Data handling
  - manages the data that is passed to and from the client. Implement checks on the data, generate web pages, etc.
- Application processing layer
  - concerned with implementing the logic of the application and so providing the required functionality to end users.
- Database
  - Stores data and provides transaction management services, etc.

ROAR

Key points

- The benefits of distributed systems are that they can be scaled to cope with increasing demand, can continue to provide user services if parts of the system fail, and they enable resources to be shared.
- Issues to be considered in the design of distributed systems include transparency, openness, scalability, security, quality of service and failure management.
- Client–server systems are structured into layers, with the presentation layer implemented on a client computer. Servers provide data management, application and database services.
- Client-server systems may have several tiers, with different layers of the system distributed to different computers.

Are there any questions?