Distributed Systems

Introduction to distributed systems

TDTS04 – Computer Networks and Distributed Systems

Carl Magnus Bruhner, ADIT/IDA

LINKÖPING UNIVERSITY

Based on slides by M. van Steen and A. S. Tanenbaum,
Distributed Systems, 4th ed., distributed-systems.net

What is this module?

- Distributed systems, part of TDTS04 (and TDDE35)
- Four lectures, giving a broad overview
- Follows the book Distributed Systems (M. van Steen & A. S. Tanenbaum)
  Available for free at: https://www.distributed-systems.net/index.php/books/ds4/
- Application of fundamentals of computer networking

Who am I?

- Second year PhD student in computer science/cybersecurity at ADIT/IDA
- Study web security and privacy
  - Web certificate ecosystem (the “Web PKI”), used for TLS/HTTPS
  - User privacy, including cookies and consent
- First-time lecturer this semester with this course
- Previously involved in the TDTS04/06/11 & TDDE35 labs for many years

What have I done with the module?

- New set of slides, based on van Steen & Tanenbaum's latest companion material to the book (where you can find more details)
- Focus:
  - Cut number of slides
  - More high-level concept than details (available in book)
  - Tighter tie to the Distributed Systems book/structure
- Exam questions will be updated to match this material (i.e., slides)
- Continuous iteration: feel free to give me feedback – and ask questions 🙋

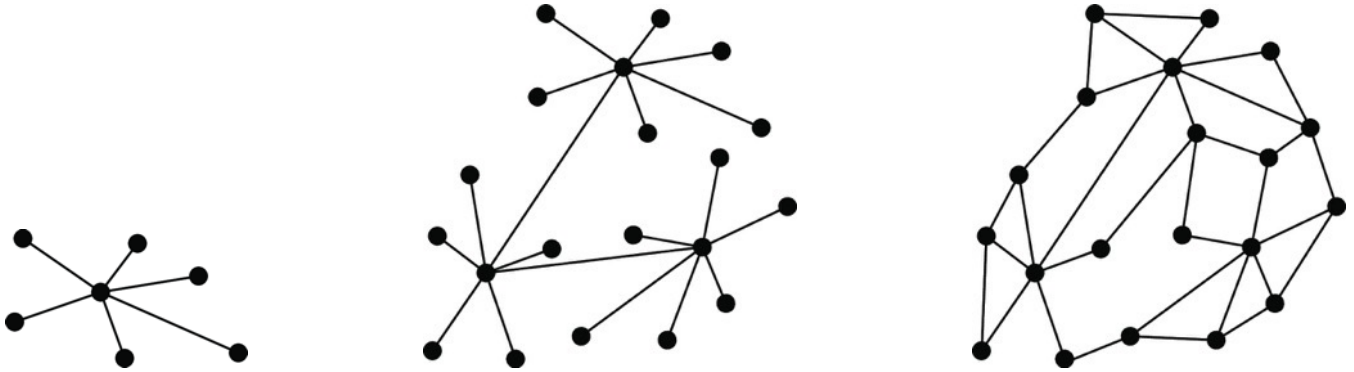
Introduction to distributed systems

Why study distributed systems?

- Understand the foundation of large-scale systems
- Understand tradeoffs when building large-scale systems
- Get knowledge useable in a broad set of applications
- Understand how the modern/connected world operates behind the scenes
- Appy knowledge of computer networking

Distributed versus Decentralized

What many people state



Centralized Decentralized Distributed

When does a decentralized system become distributed?

- Adding 1 link between two nodes in a decentralized system?
- Adding 2 links between two other nodes?
- In general: adding $k > 0$ links....?

Alternative approach

Two views on realizing distributed systems

- Integrative view: connecting existing networked computer systems into a larger a system.
- Expansive view: an existing networked computer systems is extended with additional computers

Two definitions

- A decentralized system is a networked computer system in which processes and resources are necessarily spread across multiple computers.
- A distributed system is a networked computer system in which processes and resources are sufficiently spread across multiple computers.

Recall...

Note: The following slides are based on companion material of
Computer Networking: A Top-Down Approach, 8th ed.

Thinking about the DNS

humongous distributed database:

- ~ billion records, each simple

handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters:* almost every Internet transaction interacts with DNS - msecs count!

organizationally, physically decentralized:

- millions of different organizations responsible for their records

"bulletproof": reliability, security

DNS: services, structure

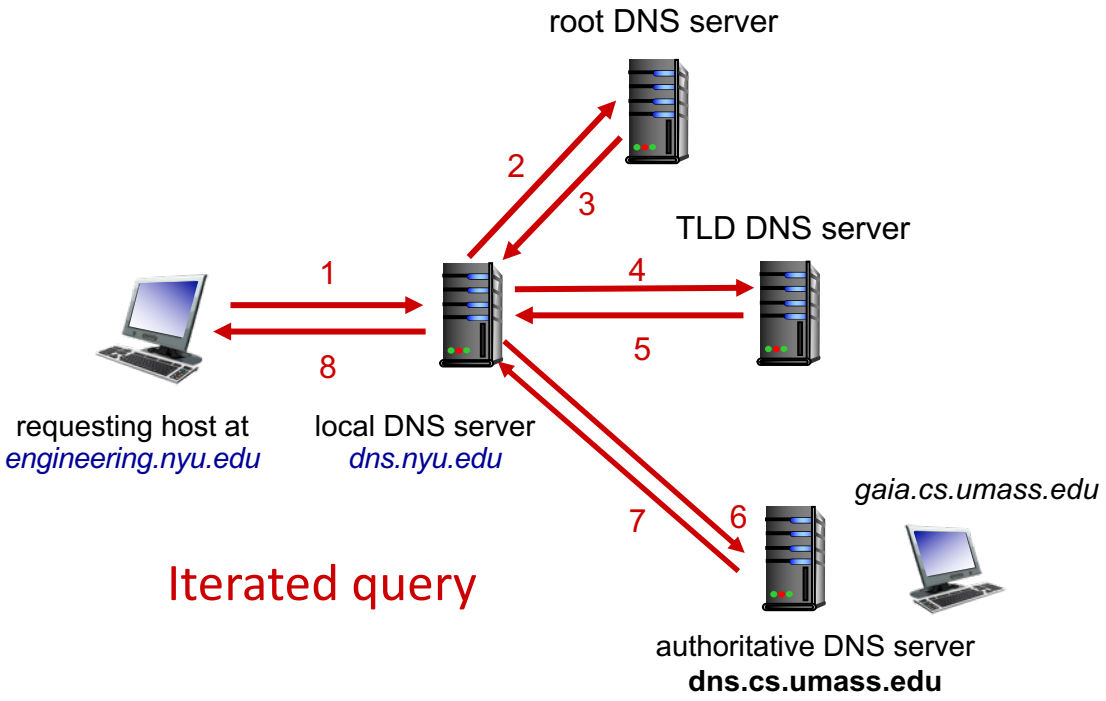
Q: Why not centralize DNS?

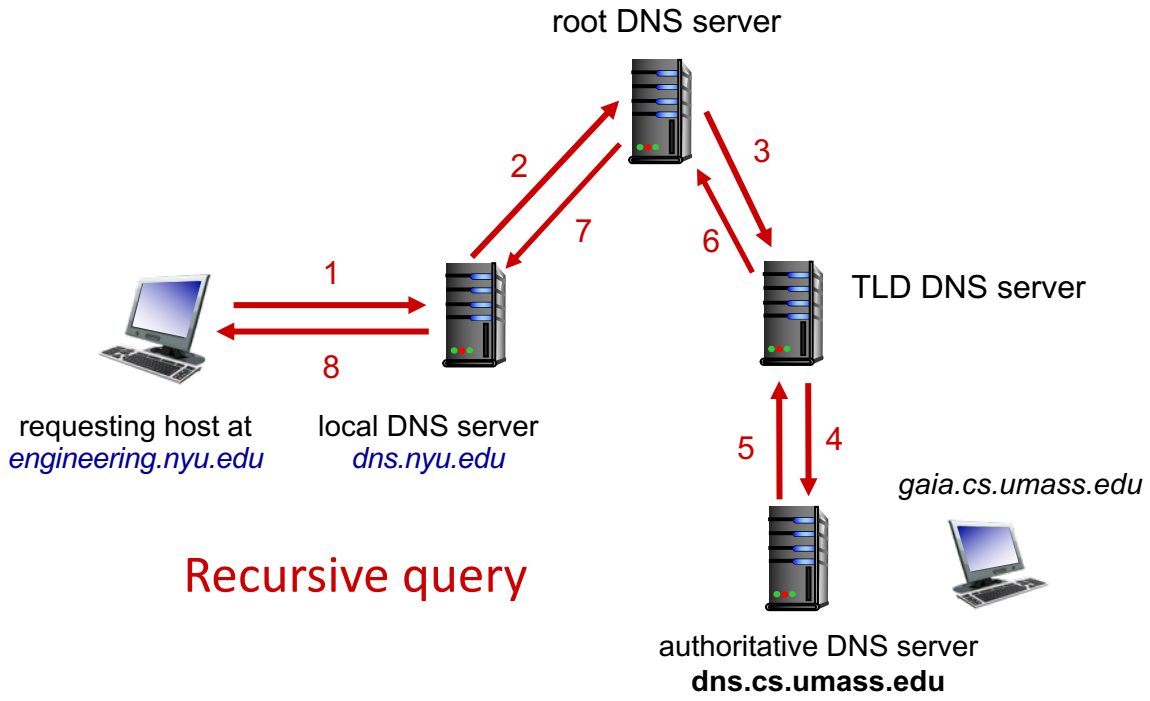
- single point of failure
- traffic volume
- distant centralized database
- maintenance

A: doesn‘t scale!

- Comcast DNS servers alone: 600B DNS queries/day
- Akamai DNS servers alone: 2.2T DNS queries/day

DNS name resolution

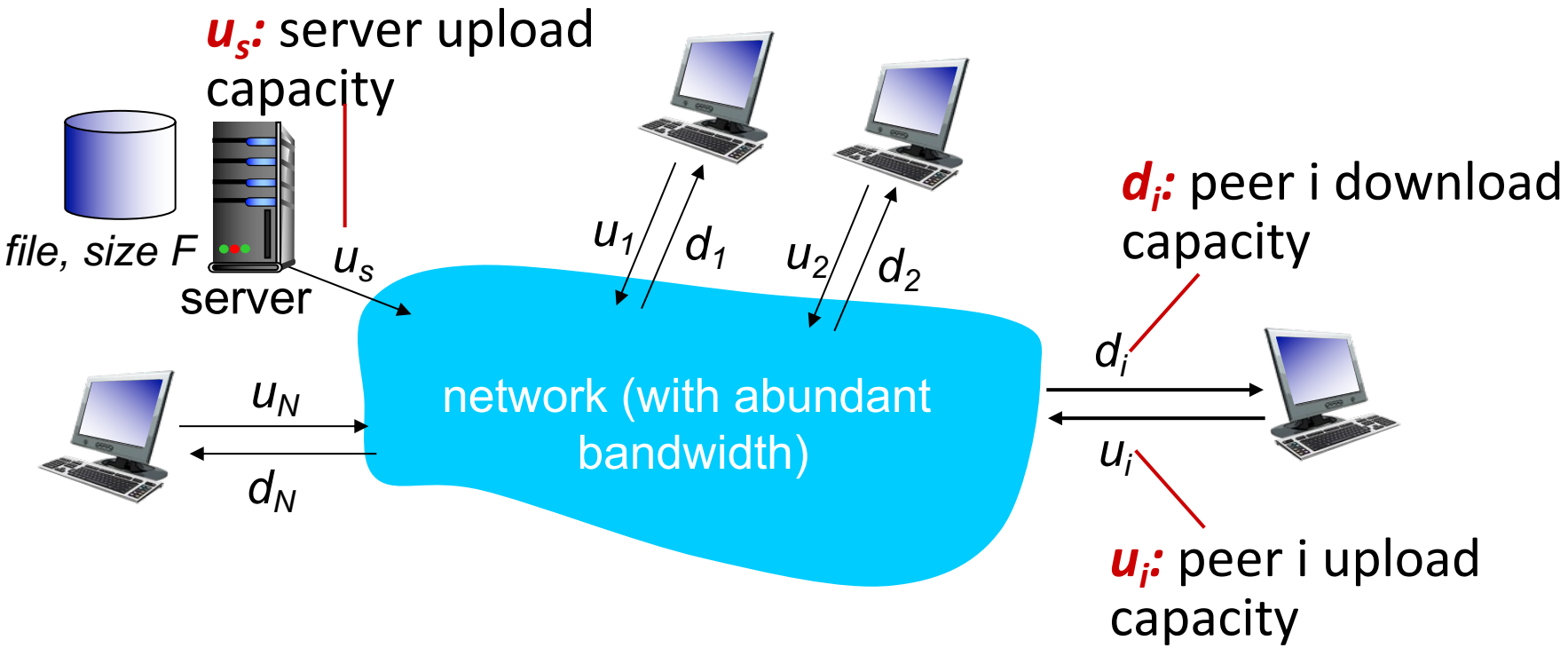




File distribution: client-server vs P2P

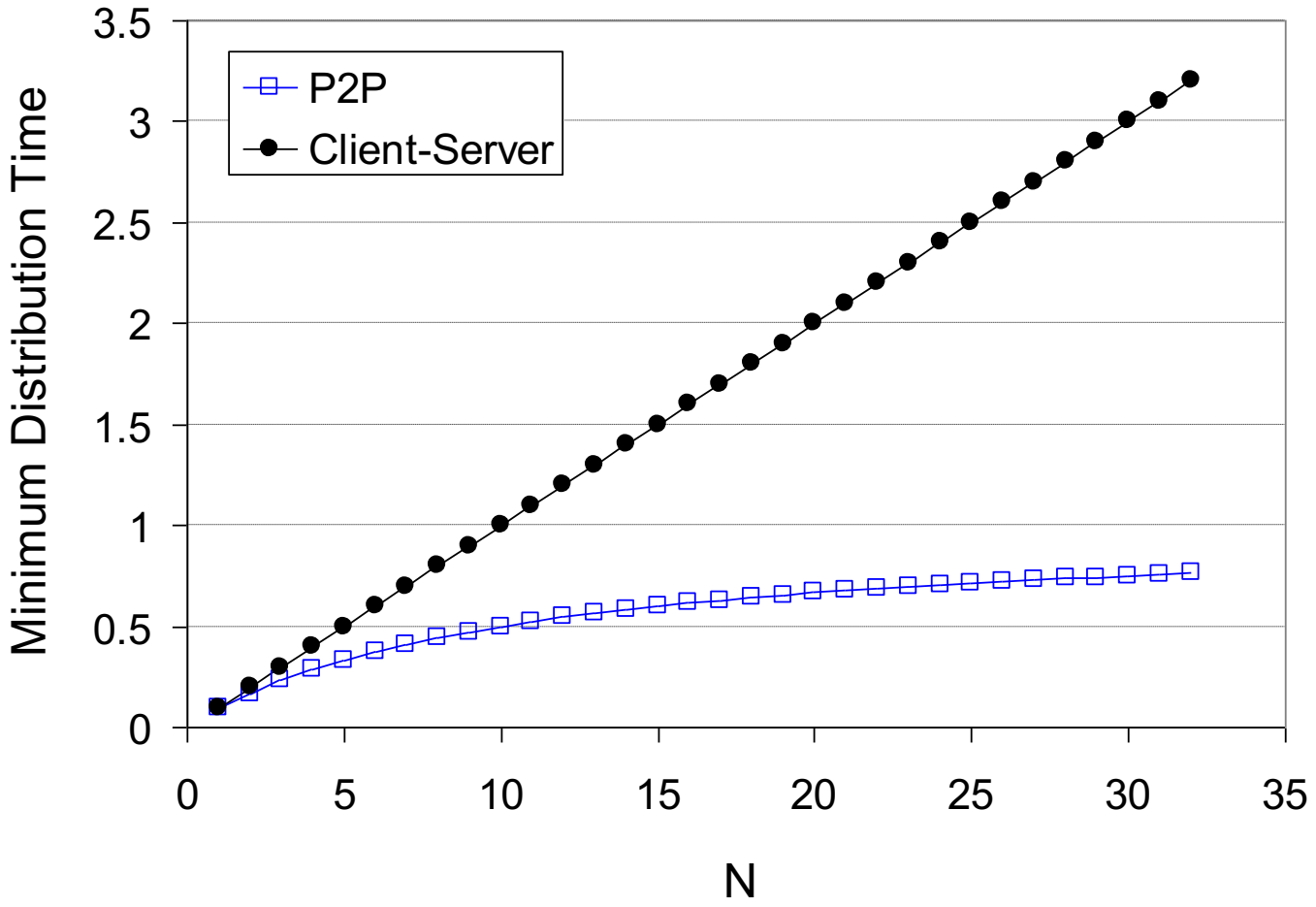
Q: how much time to distribute file (size *F*) from one server to *N peers*?

- peer upload/download capacity is limited resource



Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



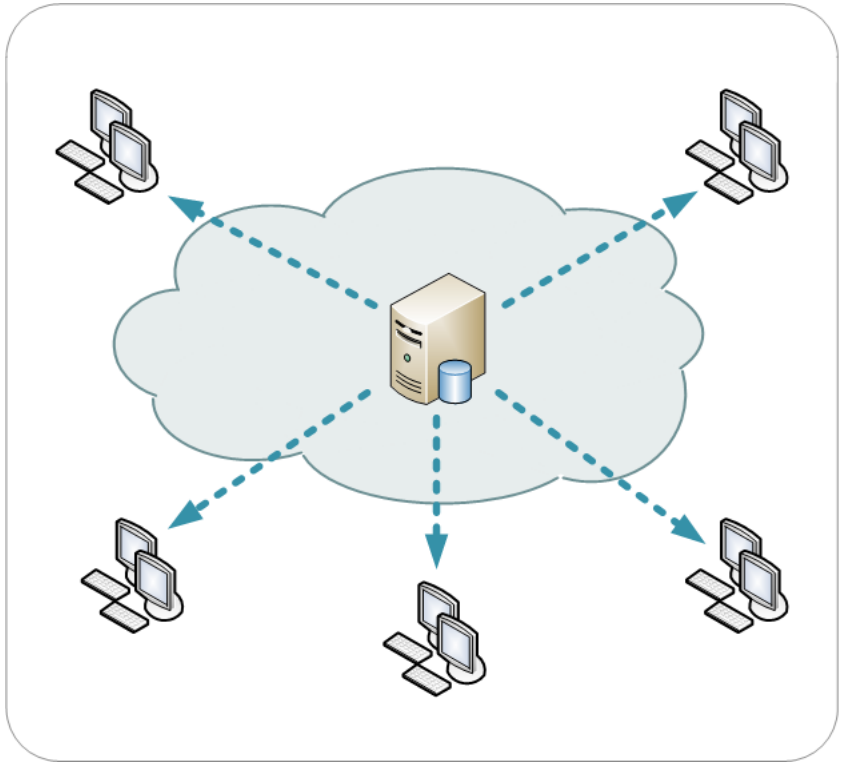
Example: Gmail


imap.gmail.com

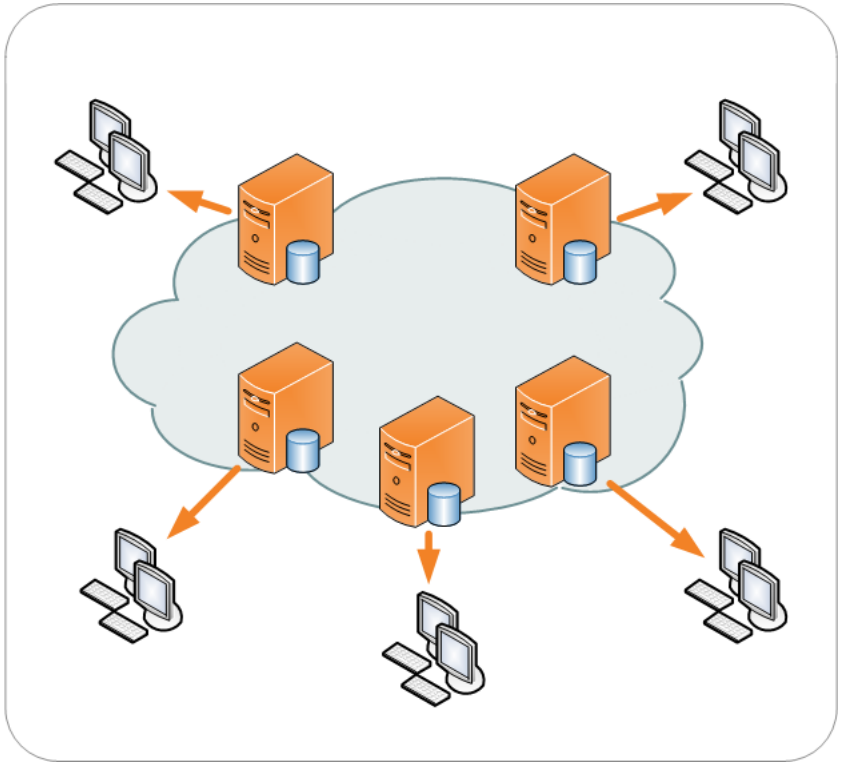

smtp.gmail.com

Gmail

Example: Content-Delivery Network (CDN)



Centralized content server
(non-CDN)



Distributed content servers
(CDN)

Back to the fundamentals…

Some common misconceptions

Centralized solutions do not scale

Make distinction between logically and physically centralized. The root of the Domain Name System:

- logically centralized
- physically (massively) distributed
- decentralized across several organizations

Centralized solutions have a single point of failure

Generally not true (e.g., the root of DNS). A single point of failure is often:

- easier to manage
- easier to make more robust

Important

There are many, poorly founded, misconceptions regarding scalability, fault tolerance, security, etc. We need to develop skills by which distributed systems can be readily understood so as to judge such misconceptions.

Perspectives on distributed systems

Distributed systems are complex: take perspectives

- Architecture: common organizations
- Process: what kind of processes, and their relationships
- Communication: facilities for exchanging data
- Coordination: application-independent algorithms
- Naming: how do you identify resources?
- Consistency and replication: performance requires of data, which need to be the same
- Fault tolerance: keep running in the presence of partial failures
- Security: ensure authorized access to resources

Perspectives on distributed systems

Distributed systems are complex: take perspectives

Lecture 2
- Architecture: common organizations
- Process: what kind of processes, and their relationships
- Communication: facilities for exchanging data

Lecture 3
- Coordination: application-independent algorithms
- Naming: how do you identify resources?

Lecture 4
- Consistency and replication: performance requires of data, which need to be the same
- Fault tolerance: keep running in the presence of partial failures

- Security: ensure authorized access to resources → *Revisited later in a separate security lecture*

What do we want to achieve?

Overall design goals

- Support sharing of resources
- Distribution transparency
- Openness
- Scalability

Sharing resources

Canonical examples

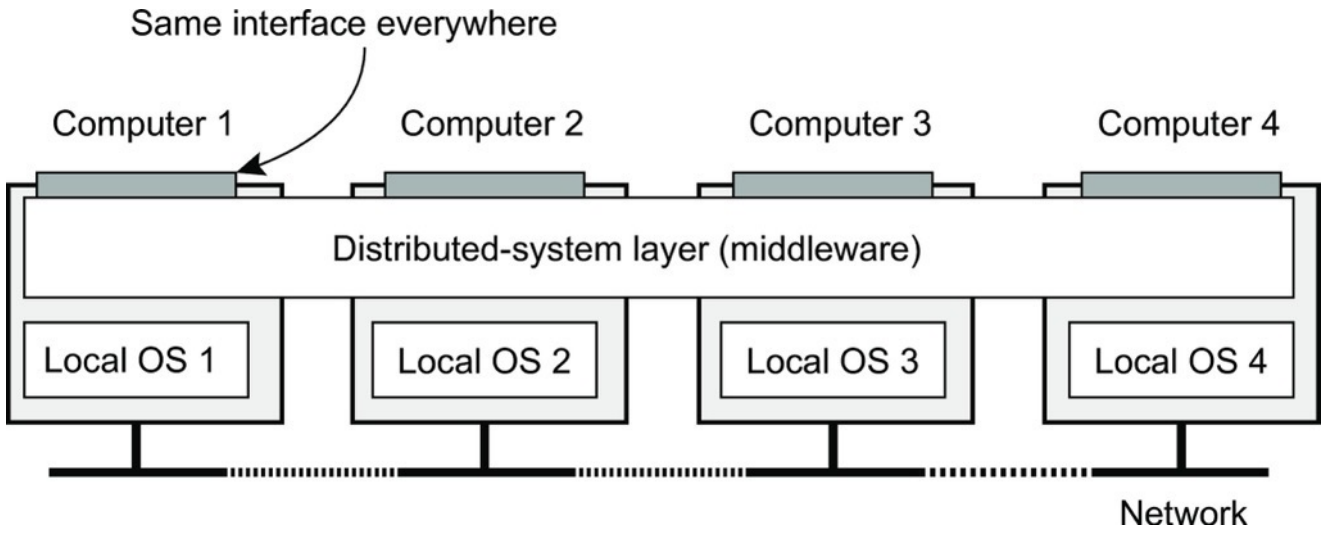
- Cloud-based shared storage and files
- Peer-to-peer assisted multimedia streaming
- Shared mail services (think of outsourced mail systems)
- Shared Web hosting (think of content distribution networks)

Observation

"The network is the computer"

(quote from John Gage, then at Sun Microsystems)

Distribution transparency



What is transparency?

The phenomenon by which a distributed system attempts to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.

Observation

Distribution transparency is handled through many different techniques in a layer between applications and operating systems: a middleware layer

Distribution transparency

Types

Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Degree of transparency

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

Degree of transparency

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 goal, but achieving it is a different story, and it should often not even be aimed at.

Openness of distributed systems

Open distributed system

A system that offers components that can easily be used by, or integrated into other systems. An open distributed system itself will often consist of components that originate from elsewhere.

What are we talking about?

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

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

Scale in distributed systems

Observation

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

At least three components

- Number of users 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. Often a solution: multiple powerful servers operating independently in parallel. Today, the challenge still lies in geographical and administrative scalability.

Size scalability

Root causes for scalability problems with centralized solutions

- The computational capacity, limited by the CPUs
- The storage capacity, including the transfer rate between CPUs and disks
- The network between the user and the centralized service

Problems with geographical scalability

- Cannot simply go from LAN to WAN: many distributed systems assume synchronous client-server interactions: client sends request and waits for an answer. Latency may easily prohibit this scheme.
- WAN links are often inherently unreliable: simply moving streaming video from LAN to WAN is bound to fail.
- Lack of multipoint communication, so that a simple search broadcast cannot be deployed. Solution is to develop separate naming and directory services (having their own scalability problems).

Problems with 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 (early versions of Skype)
- Peer-assisted audio streaming (Spotify)

Note: end users collaborate and not administrative entities.

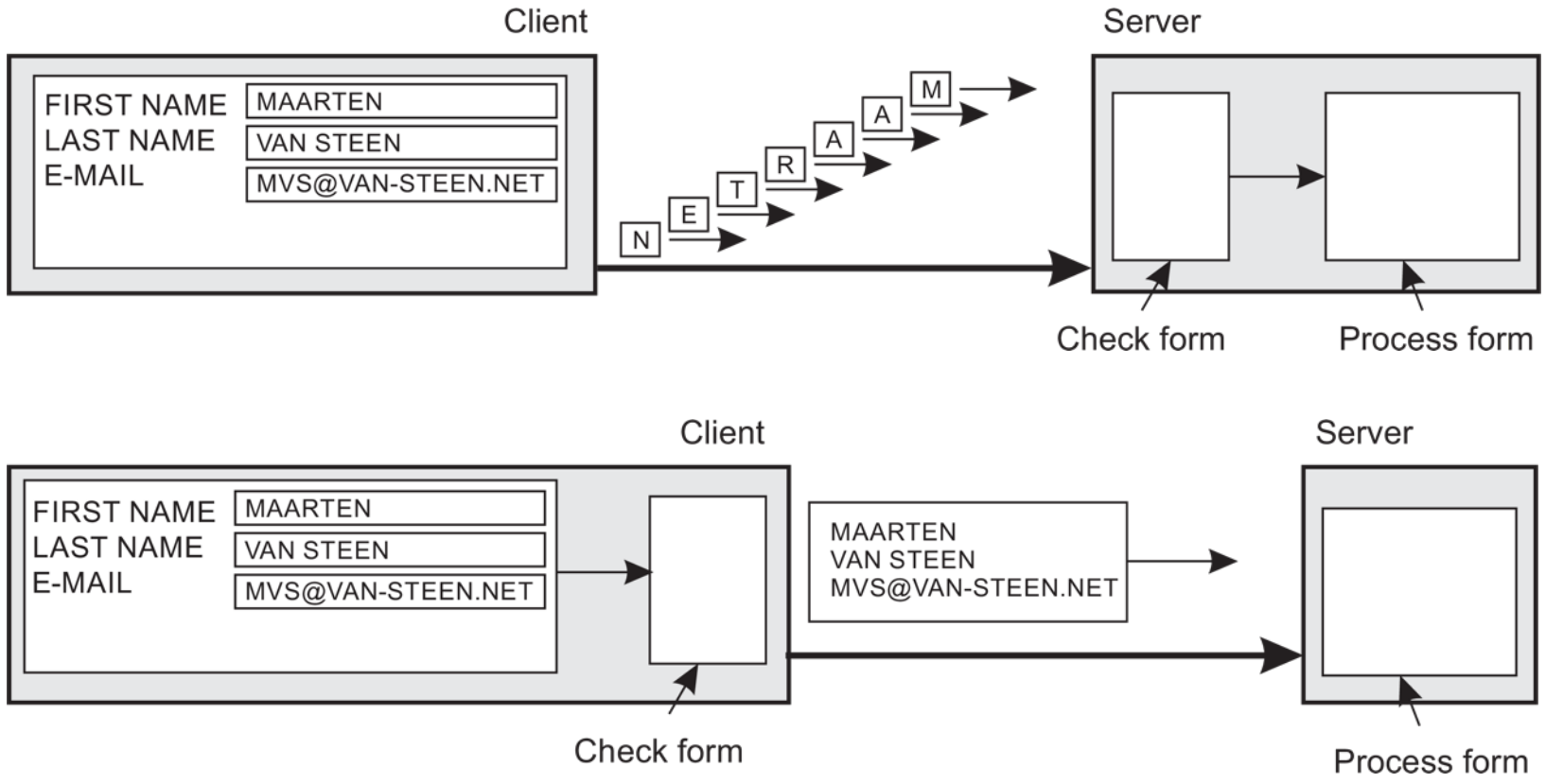
Techniques for scaling

Hide communication latencies

- Make use of asynchronous communication
- Have separate handler for incoming response
- Problem: not every application fits this model

Techniques for scaling

Facilitate solution by moving computations to client



Techniques for scaling

Partition data and computations across multiple machines

- Move computations to clients (Java applets and scripts)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)

Techniques for scaling

Replication and caching: Make copies of data available at different machines

- Replicated file servers and databases
- Mirrored Websites
- Web caches (in browsers and proxies)
- File caching (at server and client)

Scaling: The problem with replication

Applying replication 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.

How to integrate applications

File transfer: Technically simple, but not flexible:

- Figure out file format and layout
- Figure out file management
- Update propagation, and update notifications.

Shared database: Much more flexible, but still requires common data scheme next to risk of bottleneck.

Remote procedure call (RPC): Effective when execution of a series of actions is needed.

Messaging: RPCs require caller and callee to be up and running at the same time. Messaging allows decoupling in time and space.

Distributed pervasive systems

Observation

Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment.

Three (overlapping) subtypes

- Ubiquitous computing systems: pervasive and continuously present, i.e., there is a continuous interaction between system and user.
- Mobile computing systems: pervasive, but emphasis is on the fact that devices are inherently mobile.
- Sensor (and actuator) networks: pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment.

Ubiquitous systems

Core elements

1. (Distribution) Devices are networked, distributed, and accessible transparently
2. (Interaction) Interaction between users and devices is highly unobtrusive
3. (Context awareness) The system is aware of a user's context to optimize interaction
4. (Autonomy) Devices operate autonomously without human intervention, and are thus highly self-managed
5. (Intelligence) The system as a whole can handle a wide range of dynamic actions and interactions

Mobile computing

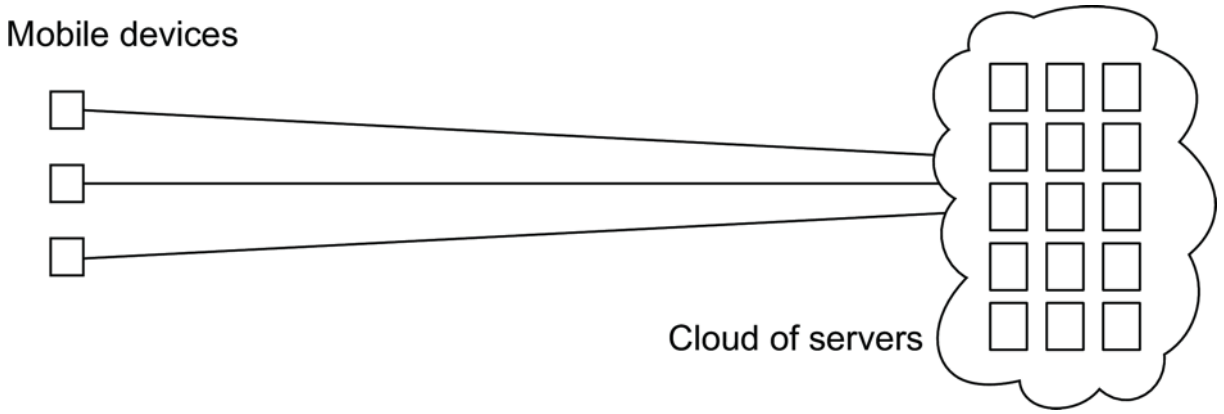
Distinctive features

- A myriad of different mobile devices (smartphones, tablets, GPS devices, remote controls, active badges).
- Mobile implies that a device's location is expected to change over time ⇒ change of local services, reachability, etc. Keyword: discovery.
- Maintaining stable communication can introduce serious problems.
- For a long time, research has focused on directly sharing resources between mobile devices. It never became popular and is by now considered to be a fruitless path for research.

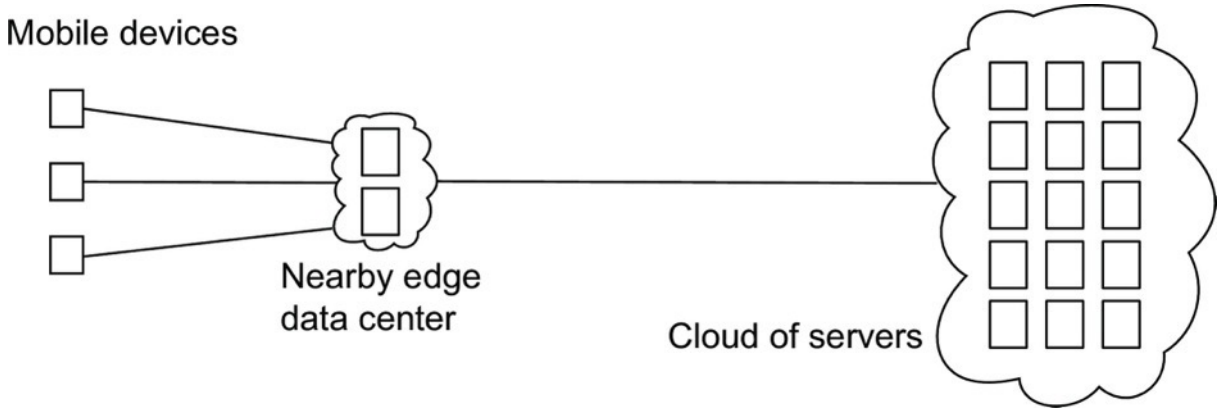
Bottomline

Mobile devices set up connections to stationary servers, essentially bringing mobile computing in the position of clients of cloud-based services.

Mobile computing



Mobile cloud computing



Mobile edge computing

Sensor networks

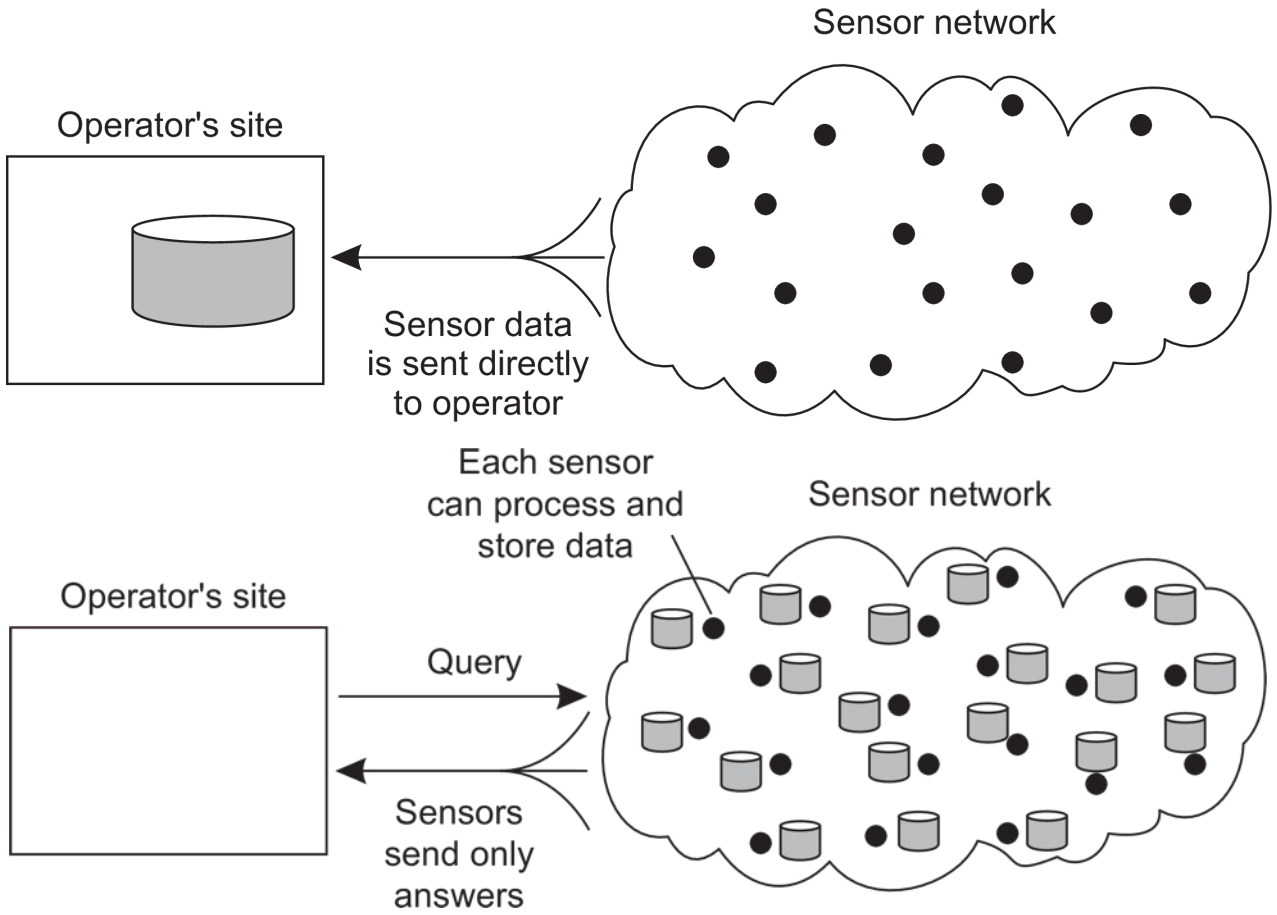
Characteristics

The nodes to which sensors are attached are:

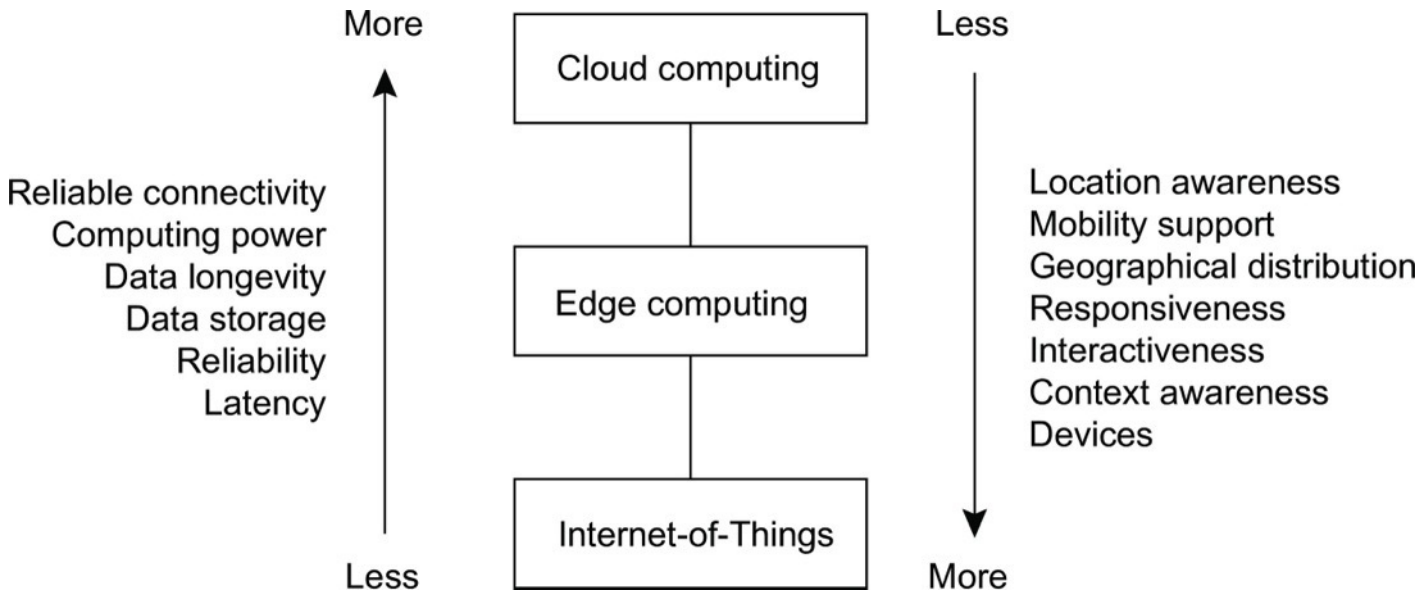
- Many (10s-1000s)
- Simple (small memory/compute/communication capacity)
- Often battery-powered (or even battery-less)

Sensor networks as distributed databases

Two extremes



The cloud-edge continuum



Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex, caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) 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

In summary

Distributed systems

- A distributed system is a networked computer system in which processes and resources are sufficiently spread across multiple computers.
- Examples include DNS, P2P, and CDNs.
- The overall design goals are sharing of resources, distribution transparency, openness, and scalability.
- Distributed pervasive systems are ubiquitous, mobile, and sensor systems supported by cloud and edge.
- Distributed systems can be very complex, with many pitfalls and risk of false assumptions.

➥ **Next lecture:** Architecture, processes, and communication

Questions?

Questions/feedback: carl.magnus.bruhner@liu.se