DISTRIBUTED SYSTEMS
(TDDB37)

Petru Eles
Institutionen för Datavetenskap (IDA)
Linköpings Universitet
email: petel@ida.liu.se
http://www.ida.liu.se/~petel
phone: 28 1396
B building, 329:220

Course Information

Web page: http://www.ida.liu.se/~TDDB37

Examination: written

Lecture notes: available from the web page, latest 24 hours before the lecture.

Text book:

Other titles can be used in addition:

Labss&Lessons:

Traian Pop
Institutionen för Datavetenskap (IDA)
email: trapo@ida.liu.se
http://www.ida.liu.se/~trapo
phone: 28 1970
B building, 3D:437

Jakob Rosén
Institutionen för Datavetenskap (IDA)
email: jakro@ida.liu.se
http://www.ida.liu.se/~jakro
phone: 28 40 46
B building, 329:224

Basic Issues

1. What is a Distributed System?

2. Examples of Distributed Systems

3. Advantages and Disadvantages

4. Design Issues with Distributed Systems

5. Preliminary Course Topics
What is a Distributed System?

A distributed system is a collection of autonomous computers linked by a computer network that appear to the users of the system as a single computer.

Some comments:

- **System architecture:** the machines are autonomous; this means they are computers which, in principle, could work independently;
- **The user’s perception:** the distributed system is perceived as a single system solving a certain problem (even though, in reality, we have several computers placed in different locations).

By running a distributed system software the computers are enabled to:

- coordinate their activities
- share resources: hardware, software, data.

☞ According to this definition, the Internet as such, is not a distributed system, but an infrastructure on which to implement distributed applications/services (such as the World Wide Web).

Examples of Distributed Systems

**Network of workstations**

- **Workstations**
- **Local area network**
- **Gateway to Wide Area Network**
- **File servers**

- Personal workstations + processors not assigned to specific users.
- Single file system, with all files accessible from specific users.
- For a certain command the system can look for the best place (workstation) to execute it.

Examples of Distributed Systems (cont’d)

**Automatic banking (teller machine) system**

- **Teller machines**
- **Bank_1 data**
- **Bank_1 backup**
- **Bank_2 data**
- **Bank_2 backup**

- Primary requirements: security and reliability.
- Consistency of replicated data.
- Concurrent transactions (operations which involve accounts in different banks; simultaneous access from several users, etc).
- Fault tolerance

Examples of Distributed Systems (cont’d)

**Automotive system (a distributed real-time system)**

- **Actuators**
- **Sensors**
- **X-by wire**
- **Input/Output**
- **PPGA**
- **CPU**
- **RAM**
- **CACHE**
- **Anti-lock breaking**
- **Adaptive cruise control**

- **Engine control**
- **Trottle control**
- **Gateway**
- **Safety critical network**
- **Non-safety critical high-speed network**
- **Non-safety critical low-speed network**
- **Entertainment network**
Examples of Distributed Systems (cont’d)

Distributed Real-Time Systems

- Synchronization of physical clocks
- Scheduling with hard time constraints
- Real-time communication
- Fault tolerance

Why do we Need Them?

Advantages of Distributed Systems

Performance: very often a collection of processors can provide higher performance (and better price/performance ratio) than a centralized computer.

Distribution: many applications involve, by their nature, spatially separated machines (banking, commercial, automotive system).

Reliability (fault tolerance): if some of the machines crash, the system can survive.

Incremental growth: as requirements on processing power grow, new machines can be added incrementally.

Sharing of data/resources: shared data is essential to many applications (banking, computer-supported cooperative work, reservation systems); other resources can be also shared (e.g., expensive printers).

Communication: facilitates human-to-human communication.

Disadvantages of Distributed Systems

Difficulties of developing distributed software: how should operating systems, programming languages and applications look like?

Networking problems: several problems are created by the network infrastructure, which have to be dealt with: loss of messages, overloading, ...

Security problems: sharing generates the problem of data security.

Design Issues with Distributed Systems

Design issues that arise specifically from the distributed nature of the application:

- Transparency
- Communication
- Performance & scalability
- Heterogeneity
- Openness
- Reliability & fault tolerance
- Security
**Transparency**

How to achieve the single system image?
How to "fool" everyone into thinking that the collection of machines is a "simple" computer?

- **Access transparency**
  - local and remote resources are accessed using identical operations.

- **Location transparency**
  - users cannot tell where hardware and software resources (CPUs, files, data bases) are located; the name of the resource shouldn’t encode the location of the resource.

- **Migration (mobility) transparency**
  - resources should be free to move from one location to another without having their names changed.

**Transparency (cont'd)**

- **Replication transparency**
  - the system is free to make additional copies of files and other resources (for purpose of performance and/or reliability), without the users noticing.
  - Example: several copies of a file; at a certain request that copy is accessed which is the closest to the client.

- **Concurrency transparency**
  - the users will not notice the existence of other users in the system (even if they access the same resources).

- **Failure transparency**
  - applications should be able to complete their task despite failures occurring in certain components of the system.

- **Performance transparency**
  - load variation should not lead to performance degradation.
  - This could be achieved by automatic reconfiguration as response to changes of the load; it is difficult to achieve.

**Communication**

Components of a distributed system have to communicate in order to interact. This implies support at two levels:

1. Networking infrastructure (interconnections & network software).

2. Appropriate communication primitives and models and their implementation:
   - communication primitives:
     - send
     - receive
     - remote procedure call (RPC)
   - communication models:
     - client-server communication: implies a message exchange between two processes: the process which requests a service and the one which provides it;
     - group multicast: the target of a message is a set of processes, which are members of a given group.

**Performance and Scalability**

Several factors are influencing the performance of a distributed system:

- The performance of individual workstations.
- The speed of the communication infrastructure.
- Extent to which reliability (fault tolerance) is provided (replication and preservation of coherence imply large overheads).
- Flexibility in workload allocation: for example, idle processors (workstations) could be allocated automatically to a user's task.

**Scalability**

The system should remain efficient even with a significant increase in the number of users and resources connected:

- cost of adding resources should be reasonable;
- performance loss with increased number of users and resources should be controlled;
- software resources should not run out (number of bits allocated to addresses, number of entries in tables, etc.)
Heterogeneity

- Distributed applications are typically heterogeneous:
  - different hardware: mainframes, workstations, PCs, servers, etc.;
  - different software: UNIX, MS Windows, IBM OS/2, Real-time OSs, etc.;
  - unconventional devices: teller machines, telephone switches, robots, manufacturing systems, etc.;
  - diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.

The solution
Middleware, an additional software layer to mask heterogeneity

Openness

- One of the important features of distributed systems is openness and flexibility:
  - every service is equally accessible to every client (local or remote);
  - it is easy to implement, install and debug new services;
  - users can write and install their own services.

- Key aspect of openness:
  - Standard interfaces and protocols (like Internet communication protocols)
  - Support of heterogeneity (by adequate middleware, like CORBA)

Openness (cont’d)

Software Architecture:

```
Applications & Services
Middleware
Operating System
Hardware: Computer&Network
```

"the platform"

Openness (cont’d)

The same, looking at two distributed nodes:
Reliability and Fault Tolerance

One of the main goals of building distributed systems is improvement of reliability.

**Availability:** If machines go down, the system should work with the reduced amount of resources.
- There should be a very small number of critical resources; critical resources: resources which have to be up in order the distributed system to work.
- Key pieces of hardware and software (critical resources) should be replicated ⇒ if one of them fails another one takes up - redundancy.

Data on the system must not be lost, and copies stored redundantly on different servers must be kept consistent.
- The more copies kept, the better the availability, but keeping consistency becomes more difficult.

**Fault-tolerance** is a main issue related to reliability: the system has to detect faults and act in a reasonable way:
- mask the fault: continue to work with possibly reduced performance but without loss of data/information.
- fail gracefully: react to the fault in a predictable way and possibly stop functionality for a short period, but without loss of data/information.

Security

Security of information resources:

1. **Confidentiality**
   Protection against disclosure to unauthorised person

2. **Integrity**
   Protection against alteration and corruption

3. **Availability**
   Keep the resource accessible

Distributed systems should allow communication between programs/users/resources on different computers.

The appropriate use of resources by different users has to be guaranteed.

Course Topics at a Glance

**Basics**
- Introduction
- Models of Distributed Systems
- Communication in Distributed Systems

**Middleware**
- Distributed Heterogeneous Applications and CORBA
- Peer-to-Peer Systems

**Theoretical Aspects/Distributed Algorithms**
- Time and State in Distributed Systems
- Distributed Mutual Exclusion
- Election and Agreement

**Distributed Data and Fault Tolerance**
- Replication
- Recovery and Fault Tolerance

**Distributed Real-Time Systems**

Course Topics

- **Introduction**
  - just finished!

- **Communication in Distributed Systems**
  - Message passing and the client/server model
  - Remote Procedure Call
  - Group Communication

- **Distributed Heterogeneous Applications and CORBA**
  - Heterogeneity in distributed systems
  - Middleware
  - Objects in distributed systems
  - The CORBA approach

- **Peer-to-Peer Systems**
  - Basic design issues
  - The Napster file sharing system
  - Peer-to-peer middleware

- **Time and State in Distributed Systems**
  - Time in distributed systems
  - Logical clocks
  - Vector clocks
  - Causal ordering of messages
  - Global states and state recording
Course Topics (cont’d)

- **Distributed Mutual Exclusion**
  - Mutual exclusion in distributed systems
  - Non-token based algorithms
  - Token based algorithms
  - Distributed elections

- **Replication**
  - Motivation for replication
  - Consistency and ordering
  - Total and causal ordering
  - Update protocols and voting

- **Recovery and Fault Tolerance**
  - Transaction recovery
  - Checkpointing and recovery
  - Fault tolerance in distributed systems
  - Hardware and software redundancy
  - Byzantine agreement

- **Distributed Real-Time Systems**
  - Physical Clocks
  - Clock Synchronization
  - Real-Time Scheduling
  - Real-Time Communication