Distributed Systems

Armin R. Mikler

Overview

- Characteristics of a DS
  - what exactly is a DS
  - what are the issues in DS
  - Distributed Systems vs. Distributed Algorithms
- Review of OS Concepts:
  - Processes
  - Process Synchronization
  - Deadlocks
  - Resource Management
  - IPC
- Time and why it matters
  - Clock Synchronization
  - Logical Clocks (Lamport)
  - Event Ordering
  - Global States*
- Distr. Mutual Exclusion
  - The naïve algorithm
  - Ricart & Agrawala
  - Maekawa's Algorithm
  - Predicate Locks in Distr. DB*

Overview cont'd

- Distr. Deadlock Detection
  - What is a distr. deadlock
  - Why is it difficult
  - Some Algorithms
    - Path Pushing
    - Edge Chasing
- Fault Tolerance
  - When things go wrong
  - Dealing with Fault
  - Reliable Transactions
    - Commit Protocols
  - Byzantine Generals
- Distr. Resource Management
  - Distributed Data Structures
  - Distributed File Systems*
  - Distributed Scheduling*
- From Clusters to Grids
  - Cluster Computing
  - Intro to Grid Computing
  - Grid Examples
- Where are we now – what can we expect next ??

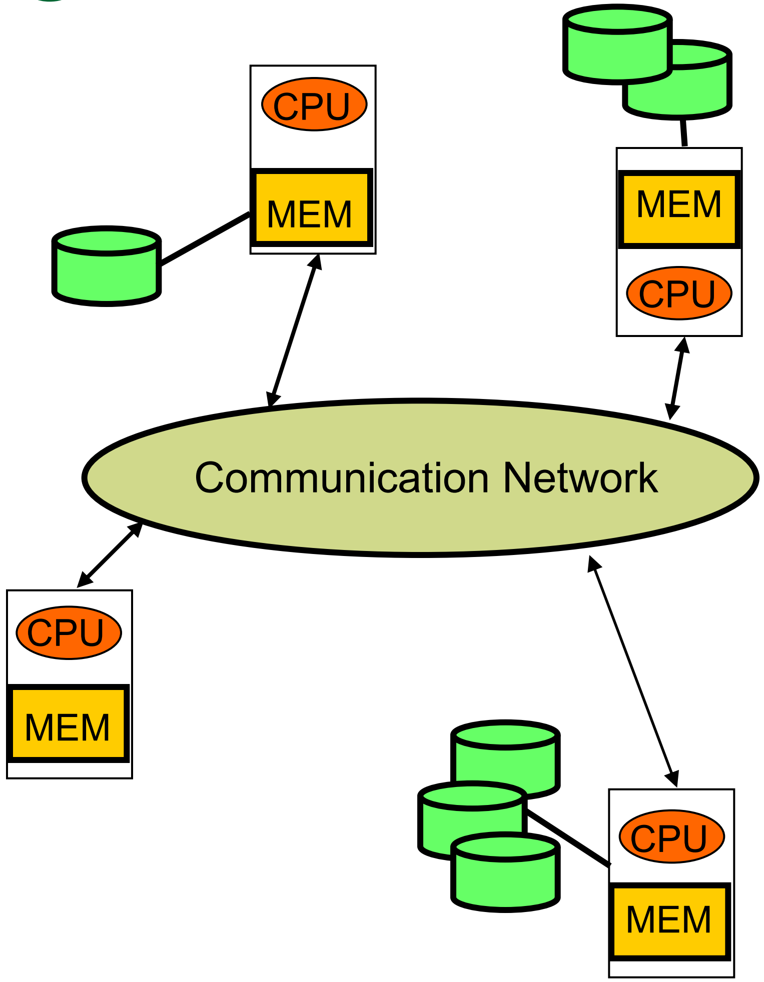
* we may skip this …

Characteristics of DS

The term *distributed system* describes a system the consists of several loosely coupled computers.

What does this mean ??

- No common or shared memory;
- No common clock;
- Computers communicate via some communication network
- Each computer has its own processor, memory, and operating system.



Motivation for DS

What are the advantages of building a DS ??

Do the advantages outweigh the associate cost and complexity of the DS ??

What are the issues that must be considered ??

In order to evaluate price/performance tradeoffs, we must decide how to measure (or express) performance of a DS !!

The advantages of DS over traditional time-sharing systems include:

<u>Resource Sharing</u>

i.e., utilization of resources from remote systems

<u>Enhanced Performance</u>

i.e., higher throughput by utilizing concurrent execution

<u>Improved Reliability/Availability</u>

i.e., intrinsic backup through duplication

<u>Modular Expandability/Scalability</u>

i.e., addition of new HW/SW without replacing existing elements.

Issues in DS

When designing a DS or the algorithms within, a number of issues must be addressed:

1. Global State / Knowledge
2. Naming
3. Scalability
4. Compatibility
5. Process Synchronization
6. Resource Management
7. Security
8. Structure

Ideally, the underlying mechanisms that address or resolve these issues should be transparent to the user / programmer.

The users view the DS as a virtual (abstracted) uni-processor, and not as a collection of individual computers.

Global State / Knowledge

In a shared memory system, the state of all processes and resources (i.e., the entire system).

Based on this global knowledge, decisions and actions among processes can effectively be coordinated.

The lack of global information, i.e., the absence of shared memory and a single globally accessible clock makes coordination of actions in DS difficult.

So, what do we need ??

1. Means to reach consensus among processes in the DS regarding the current time.
2. Means to coordinate the actions of processes based on events.
3. Means to allow for the ordering of events.

Resource Management

Resource Management in DS is concerned with making both local and remote resources available to the user.

Resource Management is roughly divided into three areas:

1. Data Migration
2. Computation Migration
3. Distributed Scheduling

The program, i.e., the location of the computation, must be able to access data.

These data is generally thought of as being contained in a file that is referenced (open, read, write, modify, etc) by the program.

Hence, one of the issues in data migration is the design of a file system that hides the location of the data w.r.t. the location of the program.

➔ Network Transparency

Resource Management cont'd

Another issue in Data Migration is shared memory access among multiple computers.

Distribute Shared Memory (SHMEM) must guarantee that the consistency of data is maintained.

Think about how SHMEM could be implemented !!

1. How should can we achieve network transparency?
2. How can we manage memory across all computers?

<u>Computation Migration:</u>

Here, the computation "moves" to another location, where the data may be available.

Paradigms for computation migration include:

- Remote Procedure Calls (RPC)
- Mobile Agents (MA)
- Process Migration

We shall look at RPC and MA in more detail !

Resource Management cont'd

The fact that jobs (or processes) may be transferred to remote machines requires a distributed scheduling strategy.

Distributed Scheduling must assure that the resources of the entire systems are utilized optimally and must assure that specific job requirements are being met.

This is of even greater importance in GRID computing environments where resources are generally owned and operated by different organizations.

Issues related to distributed scheduling include:

1. Load Balancing
2. Resource Utilization
3. Quality of Service (for Jobs)
4. Cost / Performance Tradeoff
5. Security (maybe)

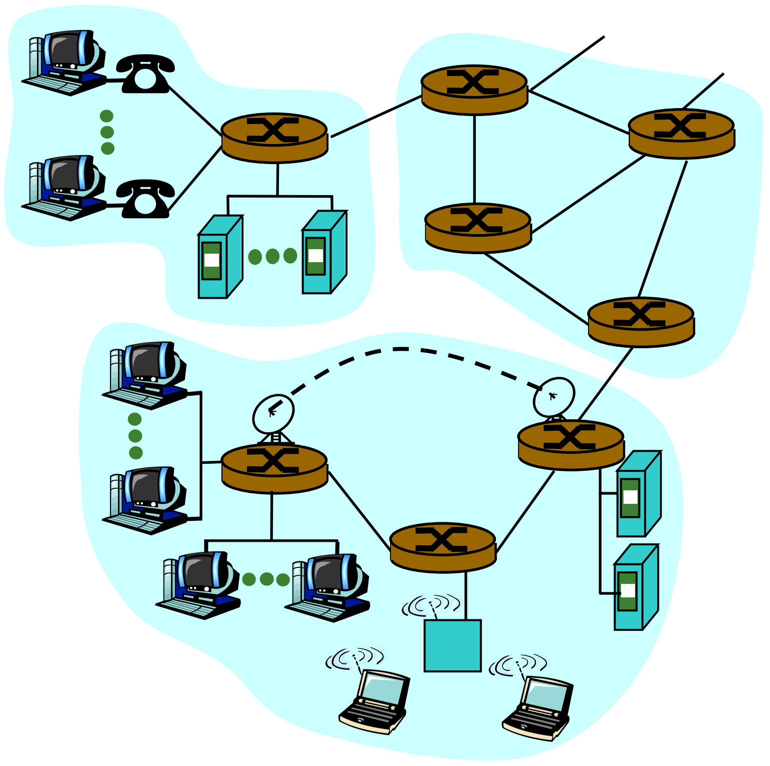
Networks and Communication

Any DS is build on top of some type of Communication Infrastructure.

Remote hosts are connected to sub-networks via switches. Subnets connect to other subnets or a backbone via routers.

We can distinguish different types of networks / network classes:

1. Wide Area Networks
2. Local Area Networks
3. Metropolitan Area Networks
4. Wireless (not really a nw-class)
5. Ad-Hoc (not really a nw-class)



DS and Networks

Recall from Computer Networks Class that all communication is governed by some communication protocol. (e.g. TCP/IP)

A DS may span as little as a small LAN or as much as multiple WANs.

Different Topologies may determine the structure of algorithms used to facilitate a DS.

1. Ring Topology
2. Star Topology
3. Tree Topology
4. Bus Topology
5. Switched subnets

The concept of topology may be extended beyond the physical layer into the logical structure of the DS.

This structure may be exploited in formulating more efficient communication sequences.

In this course, we will mostly disregard the specific structure of the network and make some simplifying assumption on reliability of the underlying communication.

Communication Primitives

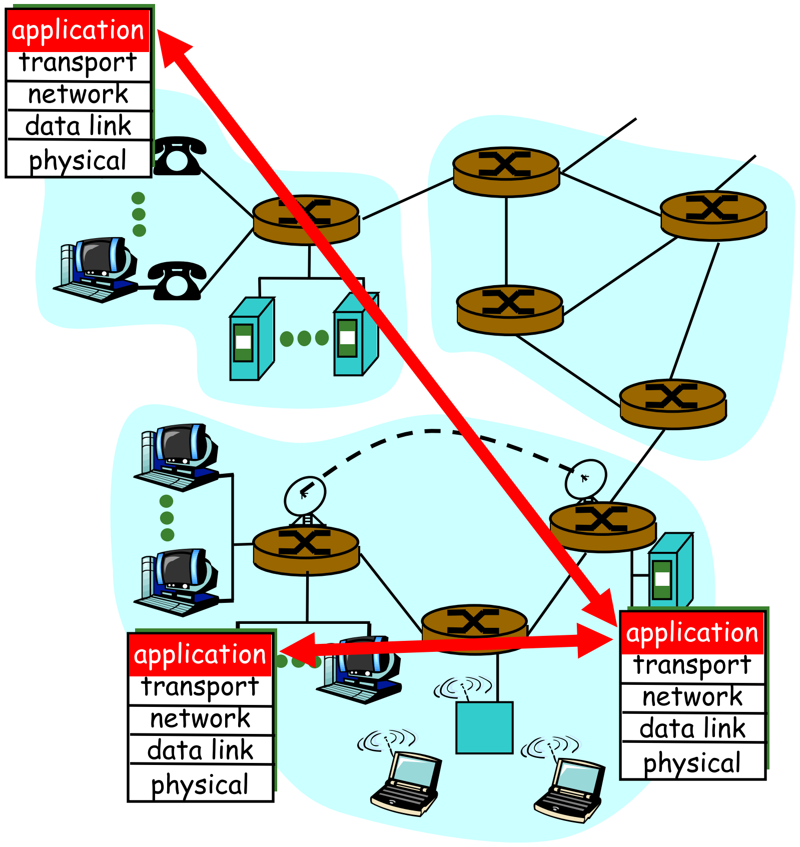
Unless the network architecture provides functions to access the lower layers of the protocol stack, communication primitives are used at the application layer.

Most DS algorithms and protocols assume only 2 primitives:

1. SEND message
2. RECEIVE message

We need to distinguish, however, two modes of primitives:

1. Synchronous
2. Asynchronous



Communication Primitives

In synchronous communication, a SEND is blocked until a corresponding RECEIVE is executed on the remote machine (or process).

With asynchronous communication, a SEND operation is buffered and the process is not blocked to wait for a corresponding receive.

What are possible implications of these semantics ??

RPC

One frequently used communication vehicle in DS is the Remote Procedure Call

RPC provides access to functionality that is available on remote hosts.

In the program, the RPC is "identical" to a local procedure call.

The fact that the procedure is actually executed remotely is hidden.

More on RPC later in this course!