KTH ROYAL INSTITUTE
OF TECHNOLOGY

Introduction

Vladimir Vlassov and Johan Montelius

What?

What is a ***distributed system***?

- Give me some examples.
- Could you give me a definition?

"…one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing."

Why?

Why do we build distributed systems?

Why?

Motivation:

- Sharing of recourses;
- To communicate;
- Data, computers and resources, and users (clients) are geographically distributed;
- To improve/achieve performance, scalability, availability, fault tolerance

Distributed applications and services

- Print servers, distributed file systems (DFS), DNS, ssh;
- WWW: web servers and browsers, FTP and mail servers/clients, instant messaging, online games, CDNs, streaming media applications, web services, etc.;
- Financial and commercial applications: E-commerce, banking (OLTP);
- Remote control and monitoring;
- Scientific and engineering computing;
- Social networks

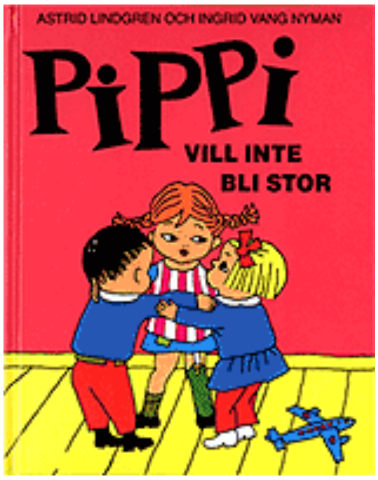
Major aspects, features and problems

- Distribution
- Concurrency
- Communication
- Messages
- Time
- Security
- Coordination
- Failures

Different from..

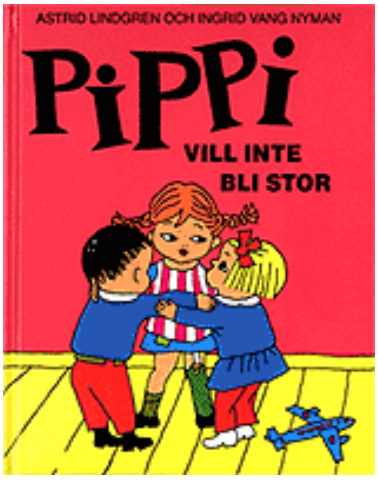
This image is of a non-distributed system – a single thread of control.

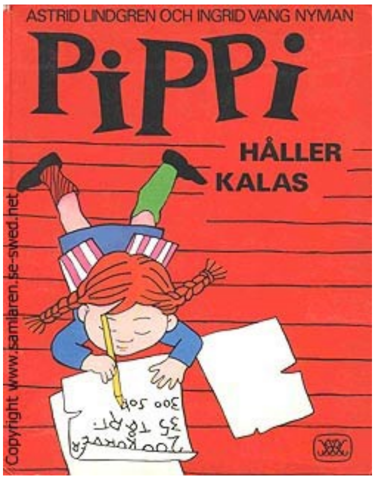
Concurrency



The first thing in a distributed system – it’s concurrent with multiple threads of control. If it’s distributed – it’s concurrent. It might be easier to communicate with shared memory. In a distributed system, there is no shared memory but only message passing – it’s a fundamental difference from a shared memory non-distributed system.

Communication





Messages

How we encode all data in a message – encoding, marshaling, un-marshaling, re-construct a data structure, how represent data, procedures, etc. – we need an infrastructure for messaging.

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of email to support: Text in character sets other than ASCII. Non-text attachments: audio, video, images, application programs, etc. Message bodies with multiple parts.

Time

How to synchronize our clocks – it's a complex problem. In one computer, synchronization is not a problem as we have ONE clock -- even in a multithreaded system. Threads can agree on the time, compare time, enforce order, etc. In a distributed system, we cannot synchronize clocks unless we have an underline synchronized network, which is unrealistic.

Security

Encrypt messages... Security: when it comes to the Internet. E.g., Distributed file system. Authentication, authorization. Access rights… We'll not talk about security in this course.

Coordination

Coordination is problematic in distributed systems. This is a two generals problem (or lunchtime problem). Agree on a time for the attack at 7:00 by sending pigeons… lunch at noon by sending emails: what would you send and respond to? E.g., “lunch noon?” – “yes, OK” – will you go to the restaurant? Assume email is reliable. Did your partner really read the reply? When? You should say “Yes, OK, please confirm” – “Yes, I confirm” 3-message system. It might be impossible to meet the deadline even if the messages are NOT LOST. We do not know how long it takes for a message to be delivered/read. – it’s a fundamental problem.

Failure

ID2201 DISTRIBUTED SYSTEMS / INTRODUCTION 13

We need to know if an interactive thread is dead or alive. If the first thread is dead, another should take over. This might be easier to achieve in a non-distributed system (through OS support). Still, In a distributed system, we might need a special process/service to monitor and inform others on failures or via heart-beat signals (say every 10 sec.) to do it over a distance. Did we solve the problem? But we have a network latency. We have a dilemma – how can we reliably detect that a thread is dead? When if we missed one hart-beat? How long shall we wait? This is a fundamental problem -- how can we reliably detect failures? When you discover that the thread is alive, you reconcile – merge. Some systems can live with this; some cannot.

Handle the problems

- How can we solve
  - communication,
  - security and,
  - coordination,

. . . in a world with failure and no notion of time?

Can we hide all problems?

What are the main differences between distributed and non-distributed systems? What are the main features of a distributed system w.r.t. non-distributed? Concurrency? Communication? Time – YES! One of the main differences is when two processing cannot agree on the time (real-time). Coordination – YES! Failure detection – YES! Sooner or later, you will see similar problems in many-core processors with on-chip networks: how long will a message take to arrive? It’s not predictable any longer.

Basic Architectures of Distributed Applications

- Two-tier architecture (a.k.a. client-server architecture):
- Three-tier architecture
- Peer-to-peer (P2P) architecture
- Service-Oriented Architecture (SOA)

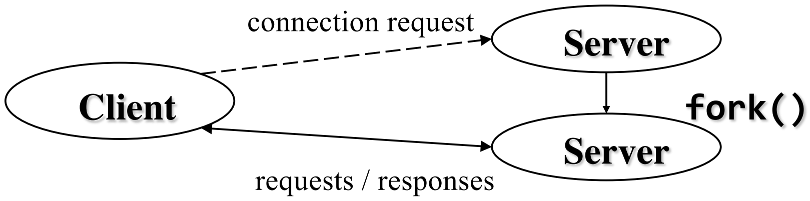
2-Tier Client-Server Architecture

The most commonly used model for distributed applications

- Can be applied for a particular request-response interaction

The ***client*** is the entity (process) accessing the remote resource, and the ***server*** provides access to the resource.

Request / response protocols



Problems of 2-Tier Client-Server on the Internet

- Portability
- Efficiency and scalability
- Fault tolerance (single point of failure)
- Security

3-Tiered Architecture

User-Interface Tier

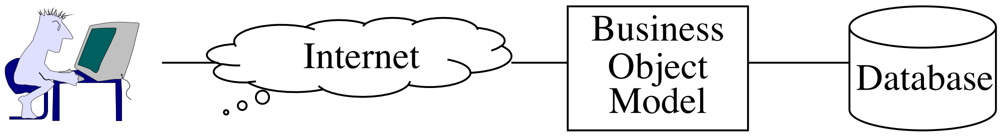
- The layer of user interaction.

Business Logic Middle-Tier

- The business logic layer comprises business objects: inventory control, budget, transaction monitors, ORBs, authentication, etc.

System Service Tier (e.g., persistent storage)

- Objects that encapsulate database routines and interact with DBMS.



3-Tier Internet Architecture Benefits

Improved performance

- Use faster protocols than HTTP
- Download a “thin” client (GUI), but leave the rest of the logic on the server or in the middle-tier

Improved scalability and fault tolerance

Manage security

- The middle tier can control user authentication and authorization w.r.t. to resources in the third tier.

Manage user application context

- The server can keep user data
- The user can access his context from any Web client

Peer-to-Peer (P2P) Architecture

A P2P system is built of ***peers*** that run on an overlay network

- All peers are ***equal*** in terms of responsibility, capabilities, and functionality

An ***overlay network*** is a "virtual" network of nodes created on top of an existing network, e.g., the Internet.

- Each node has an ID, knows neighbors, does not know the global topology, communicates as a source and a destination, and serves as a router sending data.
- Can provide a ***Distributed Hash-Table (DHT)***

Structured overlay (P2P) networks

- E.g., Chord, Pastry, Tapestry, DKS

Unstructured overlay networks

- E.g., Gnutella

General Design Issues of Distributed Systems

- Quality
  - *Functional requirements* – what it does: functions, usage scenarios, use cases, APIs.
  - *Non-functional requirements* – how good it is: performance, scalability and elasticity, complexity, availability, fault-tolerance, consistency
- Communication latency
- Failures
- Replication and Consistency
- Dynamicity (in infrastructure, resources, workload, etc.)