Networks and Distributed Systems

Introduction to Distributed Systems Design

Dr Mirco Musolesi
[based on Coulouris et al's book]

Software and hardware layers

Applications, services

Middleware

Operating system

Computer and network hardware

Platform
Software and hardware layers

- Application services and middleware can be mapped to the application layer discussed before
- Operating systems usually implement transport and network layer mechanisms
- The network hardware usually implement data link mechanisms (e.g., Ethernet, WiFi) over physical links
Communication paradigms

- Three main types of communication paradigms:
  - Inter-process communication (e.g., sockets)
  - Remote invocation
    - Request-reply protocols (e.g., HTTP)
    - Remote procedure calls (e.g., RPC Birrell Nelson 1984)
    - Remote methods invocation (e.g., Java RMI)
  - Indirect communication
    - Publish-subscribe systems
    - Distributed shared memory

Synchronous distributed systems

- Synchronous distributed systems are defined by the following bounds:
  - Bounds on execution speed (the time to execute each step of a process has known lower and upper bounds)
  - Bounds of message transmission delays (each message transmitted over a channel is received within a known bounded time)
  - Bounds on drift rates (the drift rate of the clock has a known bound)
    - Clock drift rate refers to the rate at which a computer clock deviates from a perfect reference clock
Asynchronous distributed systems

- Asynchronous distributed systems are ones which there are no bounds on:
  - Process execution speed
  - Message transmission delays
  - Clock drift rates
- Actual distributed systems are often asynchronous because of the need of the processes to share the processors and the communication channels.
Real time ordering of events

• We are often interested in knowing whether an event (sending or receiving a message) at one process occurred before, after or concurrently with another event at another process
  – Possible solution: include a timestamp in each message
• However, we have clock drifts: clocks cannot be synchronised perfectly across a distributed system:
  – Possible solution: logical clocks [Lamport 1978]

Scalability

• Distributes systems must operate at different scales from small networks to the Internet
• A system is defined as **scalable** if it will remain effective when there is a significant increase of resources and the number of users
• Challenges:
  – Controlling the cost of physical resources
  – Controlling the performance loss
  – Preventing software resources running out
  – Avoiding performance bottlenecks
Transparency

- Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system.
- The system is perceived as a whole rather than a collection of independent components.
- Various methods for implementing transparency.

Transparency

- Different types of transparency:
  - Access transparency (local and remote resources accessed using identical operations)
  - Location transparency (resources accessed without the knowledge of their location)
  - Concurrency transparency (different devices using the same resources without interferences)
  - Failure transparency (concealment of faults)
  - Replication transparency (multiple copies of resources created for increasing performances and reliability without the knowledge of users)
  - Mobility transparency (resources can be moved without the knowledge of users)
Failure models

- In a distributed system both processes and communication may fail
- A failure is a departure from what it is considered a correct behaviour
- Three types of failures:
  - Omission failures
  - Arbitrary (or byzantine) failures
  - Timing failures

Omission failures

- Two types of omission failures:
  - **Process omission failures**
    - Fail-stop failures
    - Crash failures
  - **Communication failures**
    - Send omission failure
    - Channel omission failure
    - Receive omission failure
Process omission failures

• Process omission failures are due to the fact processes halt
  – Key problem is crash detection
• Two types of omission failures based on the detection (or not) of crashes
  – Crash omission: process halts and remains halted but other processes can detect this state
  – Fail-stop omission: process halts and remains halted but other processes are not able to detect this state

Communication omission failures

Please note: this is a simplified version of the world!
Communication omission failures

- Let us consider the communications primitives send and receive
- A process $p$ performs a send by inserting the message $m$ in the outgoing message buffer
- The communication channel transport $m$ in its outgoing message buffer
- Process $q$ performs a receive by taking $m$ from its incoming message buffer and delivering it

Communication omission failure

- Three types (based on where the message is actually dropped):
  - Send omission failure: a process completes a send operation but the message is not put in its outgoing message buffer
  - Receive omission failure: a message is put in a process' incoming message buffer, but that process does not receive it
  - Channel omission failure: a message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer
Omission failures

• Reliable mechanisms are in place in order to detect and prevent these failures
• Please note: this is a very generic failure model, independent from the specific implementation
  – Failures models are useful to think about the problems
• Please note: applications might have additional buffers on top of the transport (i.e., TCP) buffers; in this model we assume that the buffer is provided by the operating system and/or middleware

Arbitrary failures

• Arbitrary (or Byzantine) failures are the worst possible failures
• An arbitrary failure of process is one in which it arbitrarily omits processing steps or takes unintended processing steps
• Essentially, they indicate the case when any type of error might have occurred
  – Examples: a process might set wrong values, unexpected reply to an invocation
Timing failures

• Timing failures are applicable in distributed systems where time limits are set on process execution time, message delivery time and clock drift rate (i.e., you should satisfy timing guarantees/constraints)

• In an asynchronous distributed system, an overloaded server may respond too slowly, but we cannot call it a failure, because we do not have time guarantees

• Timing guarantees can be provided by OSs:
  – Real-time Oss (usually called RTOS):
    • LynxOS
    • RTLinux
    • VxWorks
  – Unix/Linux and Windows 7 are NOT real time OSs

Masking failures

• Failure detection is fundamental but also handling failures is of key importance

• Masking failures is a possible strategy:
  – For example, messages can be retransmitted when they fail to arrive
  – Data can be written to multiple storage units for redundancy

• Usually systems tolerate failures!
Distributed computing as a utility

• With the increasing maturity of distributed systems infrastructure, a number of companies (Amazon, Google, Microsoft, HP, Cloudera, etc.) are promoting the view of distributed resources as a commodity or utility such as water and electricity

• Focus on sharing of resources as in the past but resources provided by suppliers

Distributed computing as a utility

• The model applies to both physical and more logical services:
  – Physical resources such as storage and processing (aka cloud computing):
    • Google Cloud, Amazon Elastic Cloud, Microsoft Cloud computing, etc.
  – Software services
    • Google Apps

• Advantages: flexibility, cost reduction (Total Cost of Ownership, including hardware costs and system administrators’ salaries), updates, etc.

• Disadvantages: difficult transitions to other providers, problems related to storing data in external servers
Cloud computing

- Cloud computing promotes the idea of everything as a service:
  - Virtual and physical infrastructure
  - Software
- Paid on a per-usage basis rather than purchased
  - New business model
- Web services offer a natural implementation path for cloud computing
  - Amazon Web Services (AWS)
  - Google App Engine