Design Challenges (Distributed Systems - Client-Server System, Peer to Peer)

1) Heterogeneity

- Heterogeneity (i.e., variety and difference) applies to the following:
  - networks
  - computer hardware
  - operating systems
  - programming languages
  - implementations by different developers
- Heterogeneity can be addressed by means of:
  - Protocols (such as Internet protocols)
  - Middleware: (software layer that provides a programming abstraction)
    we have covered this during the communication models lectures and discussions

IoT is a complex, heterogeneous, distributed systems
- Enormous differences in resources across tiers
- Many programming languages, OSes, platforms, networks, …
- Specialized hardware

2) Openness

- The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways
- In distributed applications it is determined primarily by the degree to which new resource sharing services can be added and be made available for use by a variety of client programs

Open distributed applications may be extended

- at the hardware level by the addition of computers to the network
- at the software level by the introduction of new services and the re-implementation of old ones

3) Scalability

- Scalability is the system quality attribute that addresses "growth".

Of extreme importance, because of growth rates of modern scientific and business applications–Social media, Web stores, Data mining, HPC

Growth illustration

http://dstevenwhite.com/2013/02/09/social-media-growth-2006-to-2012/

Social Media Growth 2006-2012

Year	Facebook	Twitter	LinkedIn	WordPress	Tumblr	Google+	Pinterest
2006	12,000,000	1,000	8,000,000	600,000	0	0	0
2007	50,000,000	750,000	15,000,000	2,000,000	170,000	0	0
2008	100,000,000	5,000,000	33,000,000	4,300,000	1,000,000	0	0
2009	350,000,000	75,000,000	50,000,000	8,000,000	2,000,000	0	0
2010	600,000,000	145,000,000	75,000,000	11,100,000	7,000,000	0	10,000
2011	800,000,000	300,000,000	135,000,000	50,000,000	38,000,000	90,000,000	11,700,000
2012	1,000,000,000	500,250,000	200,000,000	60,830,000	86,800,000	400,000,000	25,000,000
CAGR	109.00	507.47	71.00	120.43	248.03	344.44	4,900.00

Challenge?? resources

- How to solve this problem? Not easy! What are the strategies?
  - It is difficult to predict the demand that will be put on a system years ahead
  - Over-compensating for future growth may be worse than adapting to a change when we are forced to (for instance, larger Internet addresses will occupy extra space in messages and in computer storage)

Types of Scalability (Bondi)

- A system has Load Scalability

  If it has the ability to function gracefully, without undue delay or unproductive resource consumption and contention over a range of system loads

- A system has Space Scalability

  If its memory requirements do not grow to intolerable levels as the number of items supported increases (memory increases sub linearly!)

- A system has Space-time Scalability

  If it continues to function gracefully as the number of objects it encompasses increases by orders of magnitudes

- A system has Structural Scalability

  If its implementation and standards do not impede the growth of the number of objects it encompasses

Architecture scalability

- A *scalable architecture* can accommodate changes in usage, maintaining value for money.
- Two types of parameters that are scaled:-

- usage parameters

  e.g. number of users, input size, average behavior, number of tasks, number of connected stations

- architecture parameters

  e.g. number of servers, number of links and bandwidth, topology, distribution of tasks, number of processors

  Scalability of the architecture: the extent to which architecture parameters can and must be changed to sustain a given metric under changes in usage parameters i.e., whether the architecture can be adjusted to accommodate for the changes, and how much the 'cost' will be

Example:

usage parameter: #clients, architecture parameter: #servers, metric: response time
What would this mean for the Google File System example? Think about this at your own time

Scalability Tactics

Rules of thumb (towards scalability): limit dependencies

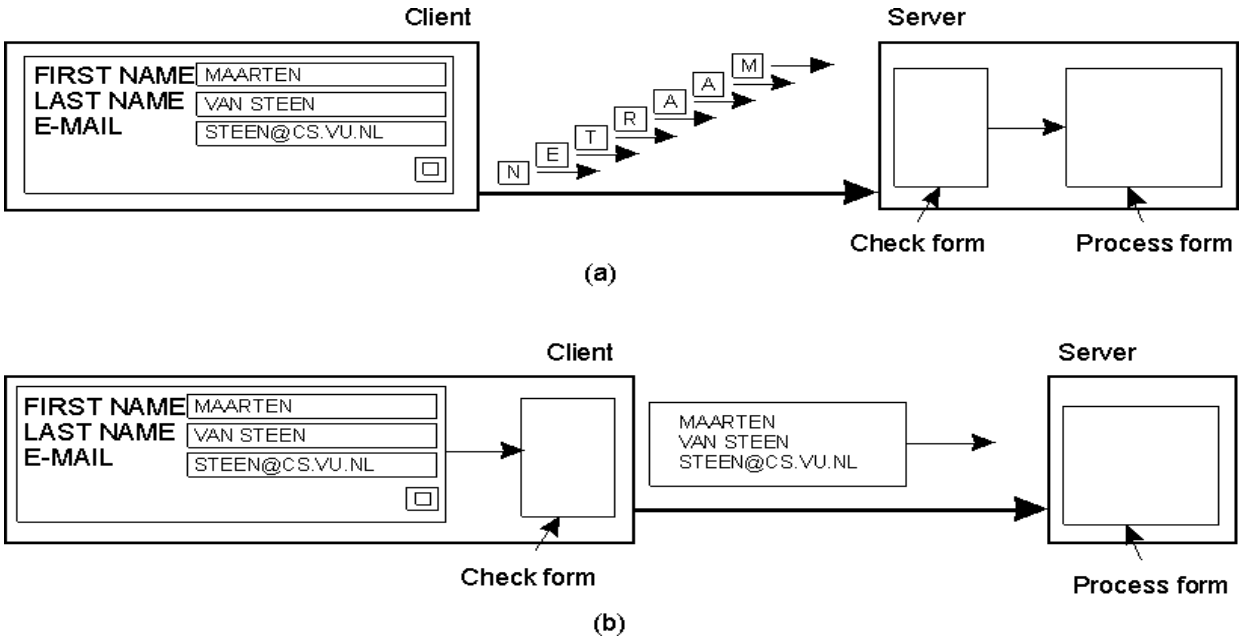
- No machine needs/has complete information
- Decisions are based on local information
- Failure of one machine does not ruin results
- No assumption of a global, shared clock

Techniques & mechanisms

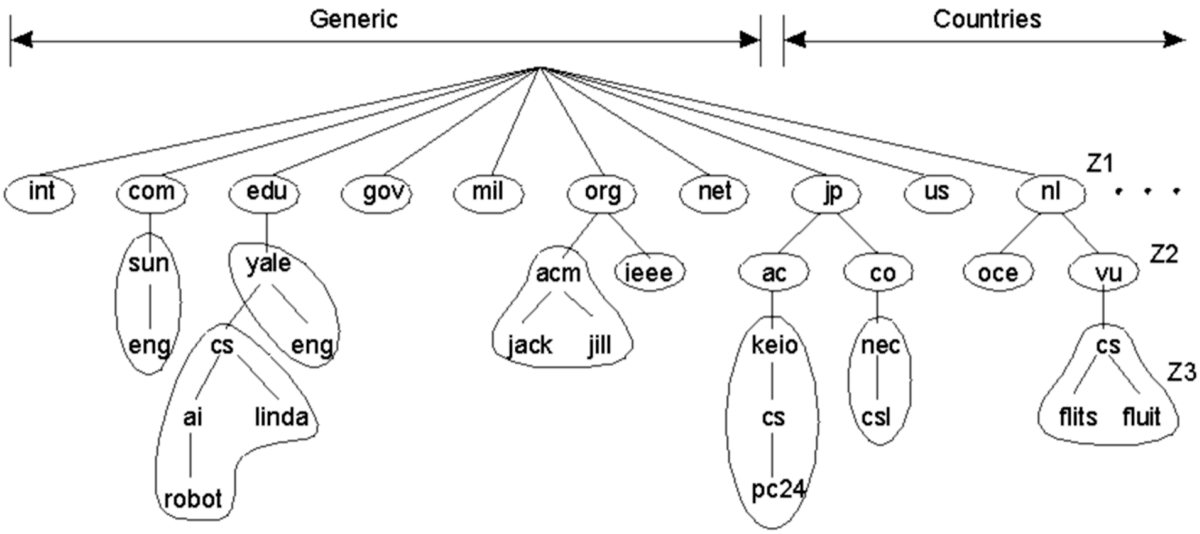
- Hide latency: do something useful while waiting
  - Asynchronous communication
  - Multi-threading
- Introduce concurrency
  - e.g. obtain parts of a webpage concurrently
- Partition, replicate or relocate data, but also work
  - bring them together (caches, map-reduce)
  - distribute evenly
  - compute same results by multiple machines
- Distribute tasks
  - using the locality principle: tasks should depend only on a small number of neighboring tasks

Scaling Techniques

- Reduce Overall communication



- Partition and Distribute



Dividing the DNS space into zones benefits from

- locality principle
- task (resolution) and data (DNS database) distribution
- information replication (caching)

Question of the architect: which architectural choices (design decisions) enhance some given extra-functional properties?

e.g. how does scalability of a client-server system compare to a peer-to- peer system? well, then we must know usage parameters, architecture parameters and metrics of choice e.g. #clients, #servers, response time and we must model the behavior and/or perform experiments

Some references on Scalability

1) Charles B. Weinstock, John B. Goodenough, *On System Scalability* , CMU/SEI-2006-TN-012, 2006
2) Leticia Duboc, David S. Rosenblum, Tony Wicks, A Framework for Characteriza-tion and Analysis of Software System Scalability, in Proc. of ESEC/FSE'07, ACM, 2007, pp 375-384.
3) A.B. Bondi, Characteristics of Scalability and Their Impact on Performance, in Proc.2nd WOSP, ACM Press, 2000, pp 195-203.

4) Failure Handling

- Computer systems *sometimes* fail
- When faults occur in hardware or software, programs may produce incorrect results or they may stop before they have completed the intended computation
- Failures in distributed systems are partial: any process, computer or network may fail independently of the others
- some components fail while others continue to function
- Therefore the handling of failures in distributed applications is particularly difficult

Failure Model

The failure model defines the way failures may occur in order to provide an understanding of the effects of failures

Omission failures: a process or communication channel fails to perform actions that it is supposed to do
Arbitrary failures: any type of error may occur
Timing failures: applicable in synchronous distributed systems

Omission Failures

Class of failure	Affects	Description
Crash	Process	Process halts prematurely and remain halted.
Omission	Channel	A msg inserted in an outgoing msg buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a send, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.

Arbitrary or Byzantine failure

- The term arbitrary or Byzantine failure is used to describe the worst possible failure semantics, in which any type of error may occur
- Arbitrary failure of a process: the process arbitrarily omits intended processing steps or takes unintended processing steps
- Communication channel arbitrary failures: message contents may be corrupted or non-existent messages may be delivered or real messages may be delivered more than once

Class of failure	Affects	Description
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behavior: it may send/ transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

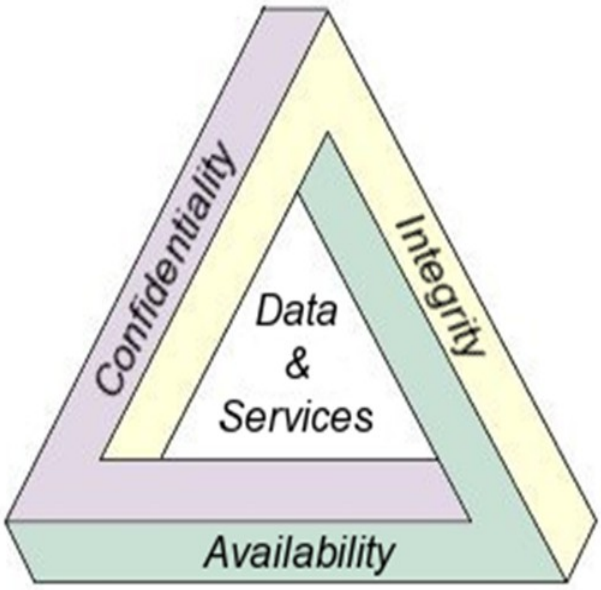
Timing failure

- Timing failures are applicable in synchronous distributed systems, where time limits are set on process execution time, message delivery time and clock drift rate
- In an asynchronous distributed system, an overloaded server may respond too slowly, but *we cannot say that it has a timing failure since no guarantee has been offered*

Class of failure	Affects	Description
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

5) Security

- Confidentiality: Protection against disclosure to unauthorized individuals
- Protection against alteration or corruption
- Protection against interference with the means to access the resources



Look at : Transparency and Consistency