Architecture of Distributed Systems 2010/2011

Distributed systems concepts & concerns

Johan Lukkien

Related to chapters 1-2: Tanenbaum & van Steen, *Distributed Systems, Principles and Paradigms*

Goals of this presentation

- Students understand extra-functional concerns guiding design choices in distributed systems.
- In particular, students understand the concept of scalability and the various forms it can take.

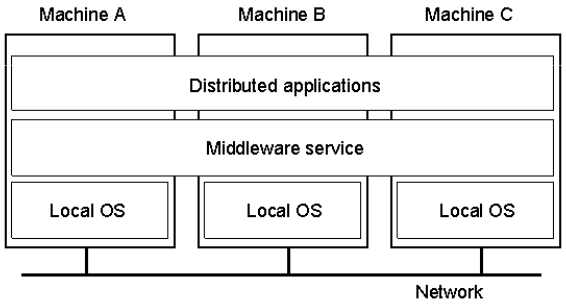
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Definition

- **Distributed system (Tanenbaum, van Steen):** a collection of independent computers that appears to its users as a single, coherent system

- **Distributed system (Lukkien):** *the hard- and software of a collection of **independent** computers that **cooperate** to realize some functionality*

- **Notes**
  - may be: serving a user, but users may not be involved directly
    - e.g. DNS, DHCP
  - independent:
    - Concurrent
    - Independent failure
    - No shared clock
    - No centralized control
    - (Spatial distribution)
  - individual parts cannot realize that functionality just by themselves



Motivation for distributed systems

- As requirement
  - distribution of the physical environment is given as part of the technical environment (a constraint)
    - different administrative domains
      - e.g. separation of data and access to data
    - ‘fact of life’ – internet connected machines
- As solution
  - introduce distribution for addressing extra-functional properties
    - concurrency – e.g performance
    - replication – e.g. reliability
    - security – e.g. separate access control from web-page access

Extra-functional aims in architectures of distributed systems

- Sharing, and independent development
  - share functionality
    - compose applications from distributed functionality
  - share resources and data
    - e.g. storage, files, centralized compute power
  - ....particularly when the distribution is a given constraint
- Transparency
  - hide internal details with respect to some viewpoint
  - ....particularly the consequences of distribution and multiple access
- Scalability
  - size: dealing with large numbers of machines, users, tasks, ...
  - location: dealing with geometric distribution and mobility
  - administration: addressing data passing through different regions of ownership
- Performance
  - by exploiting concurrency
  - balancing storage, communication, computation

Transparencies in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Scalability

- .... being able to size things up....

- Examples: *scalable algorithms, scalable networks, scalable filesystems*
  - what does: 'this parallel algorithm scales linearly' mean?
  - when is a computer network scalable? or a filesystem?

Example

- “For this discussion, file system scalability is defined as the ability to support very large file systems, large files, large directories and large numbers of files while still providing I/O performance. (A more detailed discussion of performance is contained in a later section.)”
- “By comparison, the UFS file system was designed in the early 1980’s at UC Berkeley when the scalability requirements were much different than they are today. File systems at that time were designed as much to conserve the limited available disk space as to maximize performance. This file system is also frequently referred to as FFS or the "fast" file system. While numerous enhancements have been made to UFS over the years to overcome limitations that appeared as technology marched on, the fundamental design still limits its scalability in many areas.”

- http://linux-xfs.sgi.com/projects/xfs/papers/xfs_white/xfs_white_paper.html

Scalability framework

- scale parameter, *k*
- metric, *m(k)* at scale *k*
  - measures of a quality property of the system
    - response time, network diameter, bandwidth between pairs, bisection bandwidth, reliability, utilization, cost (money)
  - scaling: compare the system at different scales: *m(k2) / m(k1)*
- use *k=1* as the reference:
  - *m*(k) = m(k) / m(1)* if *m* should be increasing
  - *m*(k) = m(1) / m(k)* if *m* should be decreasing
- scalability criterion, *S(k)*
- the system is called scalable (w.r.t. *m* and *S*) if *m*(k) ≥ S(k)*

Example: matrix multiplication

```
for i := 0 to N-1 do
  for j := 0 to N-1 do
    C[i,j] := DotProduct (A[i,*], B[*,j])
```

- Executed on a network of connected (processor, memory) pairs
- Each process(or) performs the computations required for the part of *C* it is assigned to
- To that end it needs to communicate parts of *A* and *B* stored elsewhere

- Then, an approximation of the time to execute on *P* processors is

$$T(P,N) = (2fN^3)/P + (cN^2)/P$$

  - *f* = time for a computation step
  - c = time for communication of an element
  - additional delays, including possible communication delays, ignored

Scalability metrics

- *T(P,N)* can act as a scalability metric (decreasing for *P*)
  - investigate time as function of *P* or *N*
    - *e.g. k = P: m*(k) = m(1) / m(k) = 2fN³ / T(k, N)*
    - this represents, in fact, the speedup when using *P* processors
  - somewhat unclear what the criterion should be (next slide)

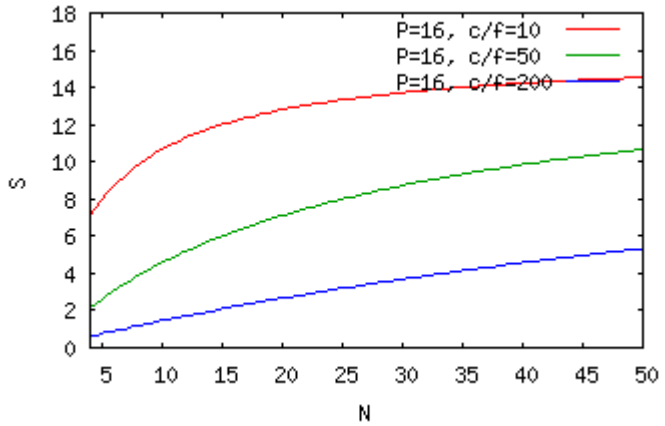
- This speedup gives the speed change in two dimensions
  - $S(P,N) = (2fN^3) / T(P,N) = P / (1+(c/2Nf))$

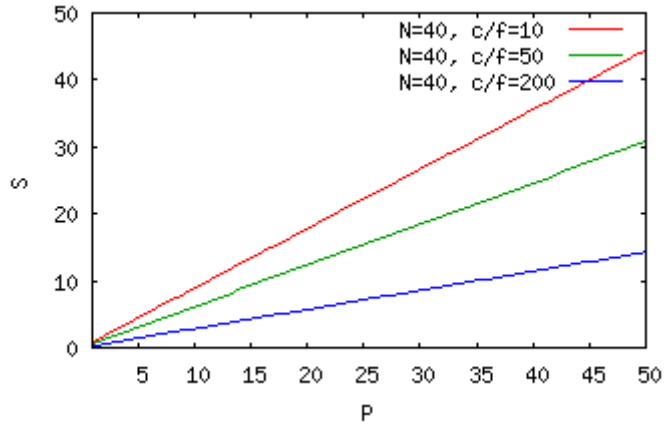
- Scalability
  - as function of *P:* how many processors can reasonably be used for a size *N* problem?
  - as function of *N:* will the speedup converge to *P* for large *N?* How fast? How large must *N* be to be close to *P*?

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Scalability

- Possible scalability criteria:
  - how large *P* can become or *N* must be while having e.g. $S(P,N) \geq 0.6P$
    - note that *S* is a factor (dependent on *N* only) times *P*
    - this is not be case in many practical realizations where smaller messages have more overhead (when $P = N^2$ each machine has just one element)
  - how fast *S(P,N)* converges to *P* as function of *N;* in particular, how fast the initial rise is (then the network may even be used for small *N*)
- Shape depends on speed of communication relative to computation
  - pictures show slow and faster communication
  - *c/f* is an important platform parameter for this system





A work preserving perspective

- We can also look at what happens if we increase work and platform (hardware) together
  - if we double the work and the investment, do we get the same efficiency?

- The value of the system *v(k)* is a metric for the systems' benefit at scale *k*
  - e.g. effective #transactions/sec, computations/sec, speedup
  - note: this is the value of work done by the considered application on a system of that scale

- The cost of the system at scale *k, C(k),* represents a cost measure for system upscaling
  - e.g. real money, # processors

- The scalability metric *m(k) = v(k)/C(k),* represents *efficiency*
  - *m*(k) = v(k)/C(k) [/ v(1)/C(1)]*

Our example

- Take the speedup as value measure
- System (work and network size) at scale *k*, cost = #processors*:*
  - $m^*(k) = S(k, k^{1/3}) / k \quad = \quad 1 / (1+ c/(2fk^{1/3}))$
- This approaches 1 from below as *k* increases
  - efficiency improves if we scale up our entire system
  - note that you might need very large problems to achieve this high efficiency on larger systems
    - however, for this example (with $N \sim k^{1/3}$) this is quite ok.
- Instead of *#processors*, we can take $$ as the cost
  - in order to have this scalability, *$$ must then rise linearly with #processors*

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Scaling, to balance usage

- Work preserving is a special case of the following:

- Two types of parameters
  - *usage* parameters
    - number of users, input size, average behavior, number of tasks, number of connected stations
  - *architecture* parameters
    - number of servers, number of links and bandwidth, topology, distribution of tasks, number of processors

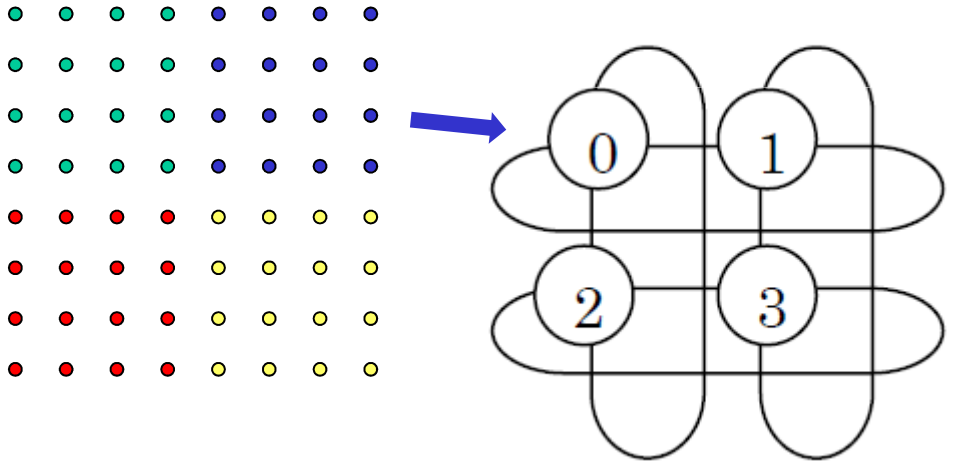
- Metrics
  - measures of quality properties
    - response time, network diameter, bandwidth between pairs, bisection bandwidth, reliability, utilization, reliability, cost (money)

- Scalability: *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

Reviewing some assumptions

- Assumptions in our example
  - communication time can be seen as an overhead, proportional to the number of elements a processor needs
  - no idle time is incurred, e.g. by waiting on communication resources
  - communication and computation don't overlap

- Possible realization:
  - block distribution of *A, B* and *C*
  - mapped on a *torus*
    - note that this architecture admits concurrency in the communication
  - circle matrix blocks around in both dimensions
  - use *latency hiding*
    - communicating while computing



Non-Scaling

- Suppose communications in the matrix multiplication are done via a shared medium
  - e.g. bus, wireless medium

- Communication overhead can then be cN^2 (without latency hiding)
  - no matter how many processors are used, this sequential term remains
    - $T(P,N) = (2fN^3)/P + (cN^2)$

    ['+' becomes 'max' with concurrency in communication and computation]

- Exercise: examine scalability metrics again

Amdahl's law

- Consider a system described with scaling parameters
  - *N* (problem size, number of jobs or clients per second) and
  - *P* (number of servers, processors)
- In order to improve performance, more machines are added
- Typically, a performance metric has two parts:
  - $m(N) = seq(N) + par(N)$
- The first term *cannot be improved* by adding more processors, which limits potential improvements
  - $m(P,N) = seq(N) + par(N)/P$
  - $S(P,N) \leq m(1,N)/seq(N)$

Amdahl's law, general lesson

- Typically, improvements focus on a part of the problem
- Whenever that part has become negligible, move your attention somewhere else

- Examples:
  - single server dispatches request
    - the server can become the bottleneck
  - improving performance of a subtask

Scalability Problems

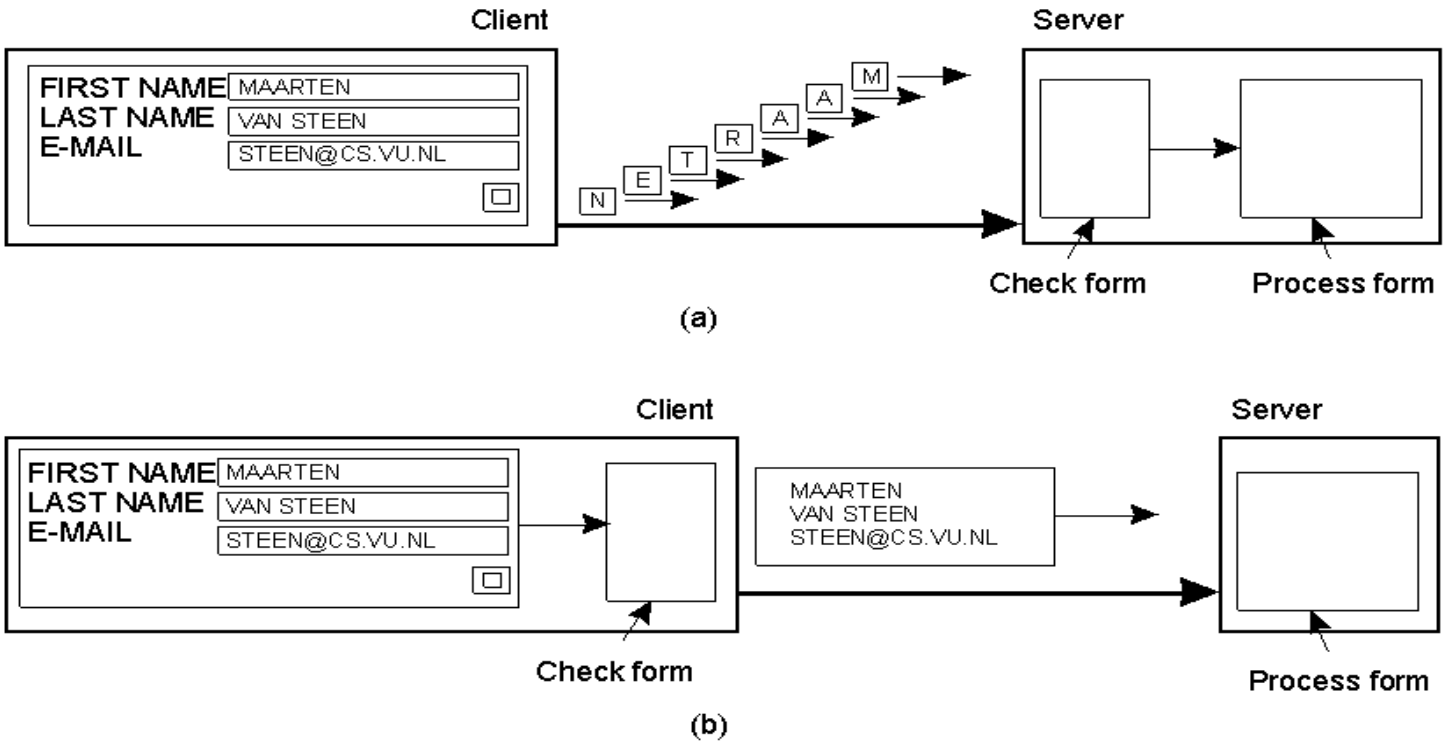
Concept	**Example**
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Scalability principles

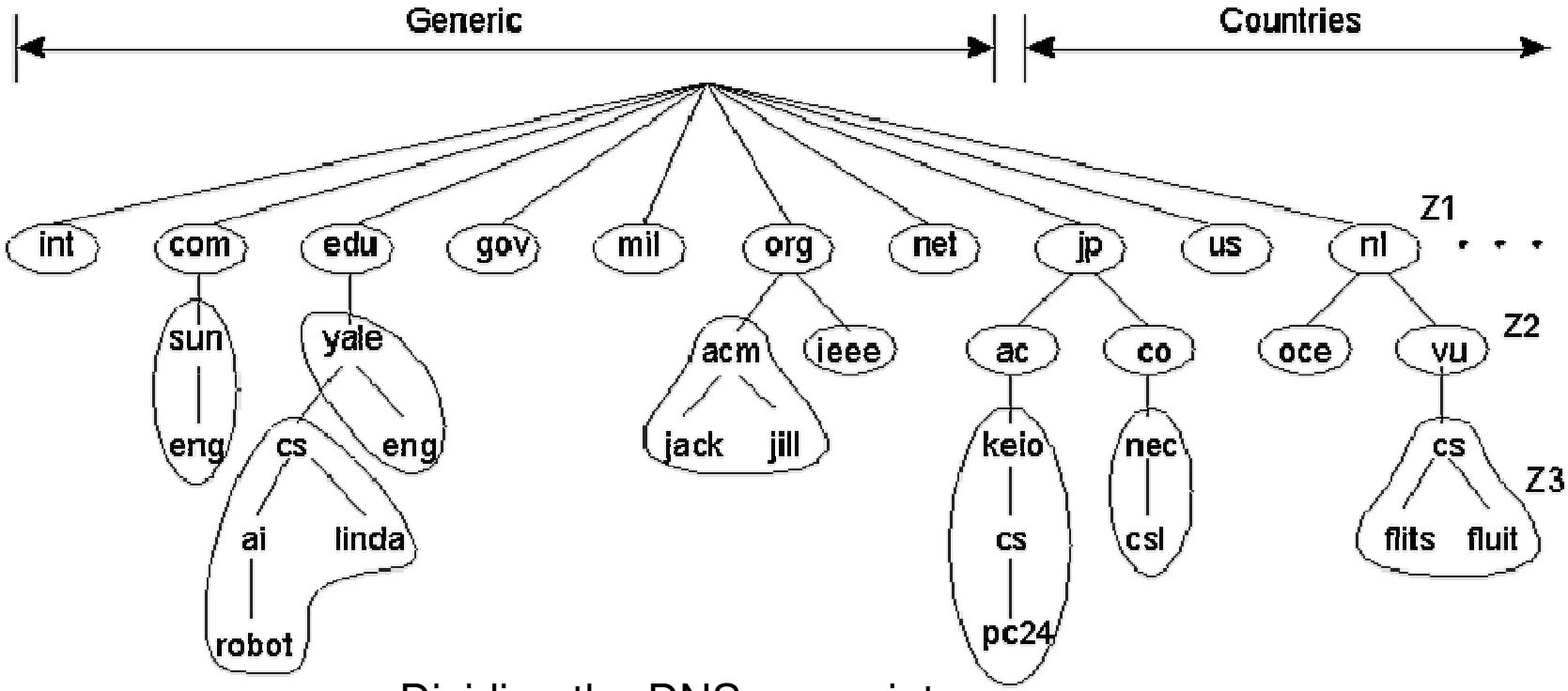
- Rules of thumb (towards scalability): limit dependencies
  - No machine needs/has complete information
  - Decisions are based on local information
  - Failure of one machine does not ruin results
  - No assumption of a global, shared clock

- Techniques & mechanisms
  - Hide latency: do something useful while waiting
  - Introduce concurrency
    - e.g. obtain parts of a webpage concurrently
  - Replicate or relocate data, but also work
    - bring them together
    - distribute evenly
  - Distribute tasks
    - using the locality principle: tasks should depend only on a small number of neighboring task

Scaling Techniques (1)



Scaling Techniques (2)



- Dividing the DNS space into zones
- Benefits from:
  - locality principle
  - task distribution
  - information replication (caching)

Extra-functional properties and architecture

- Which architectural choices (design decisions) enhance these extra-functional properties?
  - e.g. how does scalability of a client-server system compare to a peer-2-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
- This question also relates to finer choices to be made
  - e.g. the nature of the connectors (message passing, RPC, ...)
- Main views addressed in following presentations:
  - process and deployment views,
  - and logical organization of the development view