Architecture of Distributed Systems

Scalability 2018-2019

Original: J.J. Lukkien
Revision. R.H. Mak

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Goals of this presentation

Understanding the concept of scalability and the various forms it can take.

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

- There is no consensus in literature how to define scalability
  - In the context of parallel computing very precise definitions exists
  - Although helpful, these are too limited for general architectural purposes.
- Not so easily captured using scenario and tactics
  - Hardly discussed in Bass, Clements and Kazman
- Of extreme importance, because of growth rates of modern business applications
  - Social media, Web stores, etc.

Growth rates I

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

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

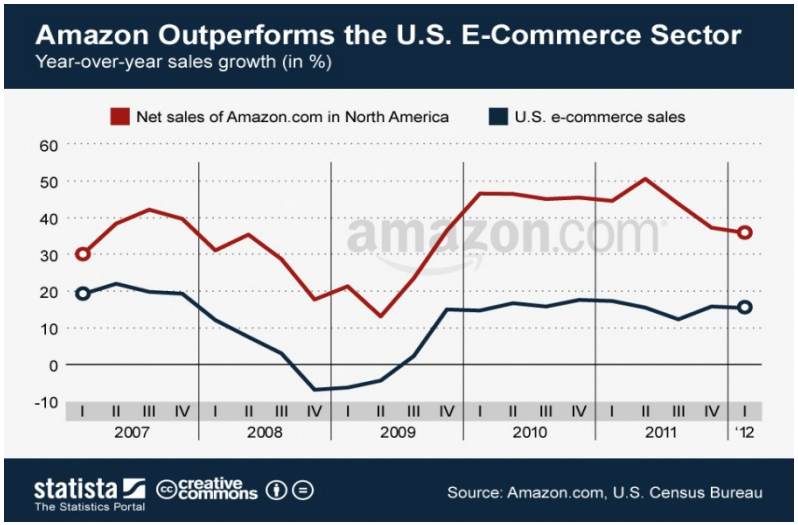
CAGR: compound annual growth rate

- Facebook: $12 * 2.09^6 \approx 1000$
- LinkedIn: $8 * 1.71^6 \approx 200$

Growth rates II

Just a few numbers

- To get an impression
- Not up to date
- Accuracy difficult to assess



http://www.statista.com/topics/846/amazon/

Conclusion.

In view of immense growth rates, scalability has to be addressed right from the beginning and not as an afterthought.

Two definitions

Weinstock & Goodenough: CMU/SEI-2006-TN-012

- **http://www.sei.cmu.edu/reports/06tn012.pdf**

Definition 1

Scalability is the ability to handle increased workload (without adding resources to a system).

Definition 2

Scalability is the ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capacity.

Example: file system scalabilty

- “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**

Types of Scalability (Bondi 2000)

- 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 sublinearly!)
- 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

Need for a scalability framework

These definitions are interesting but not "good enough", because they are

1. Not specific:
   - To become operational, "ability" has to be defined for each individual system, but this holds for any general definition.
   - More importantly, they do not provide any handles on how they can be instantiated in a systematic way.
2. Not quantitative but qualitative:
   - They cannot be used to quantify the degree of scalability, hence it is hardly possible to *compare* architectures.
   - They cannot be used to *analyze* scalability in a quantitative manner to detect, or show the absence of, architectural bottlenecks

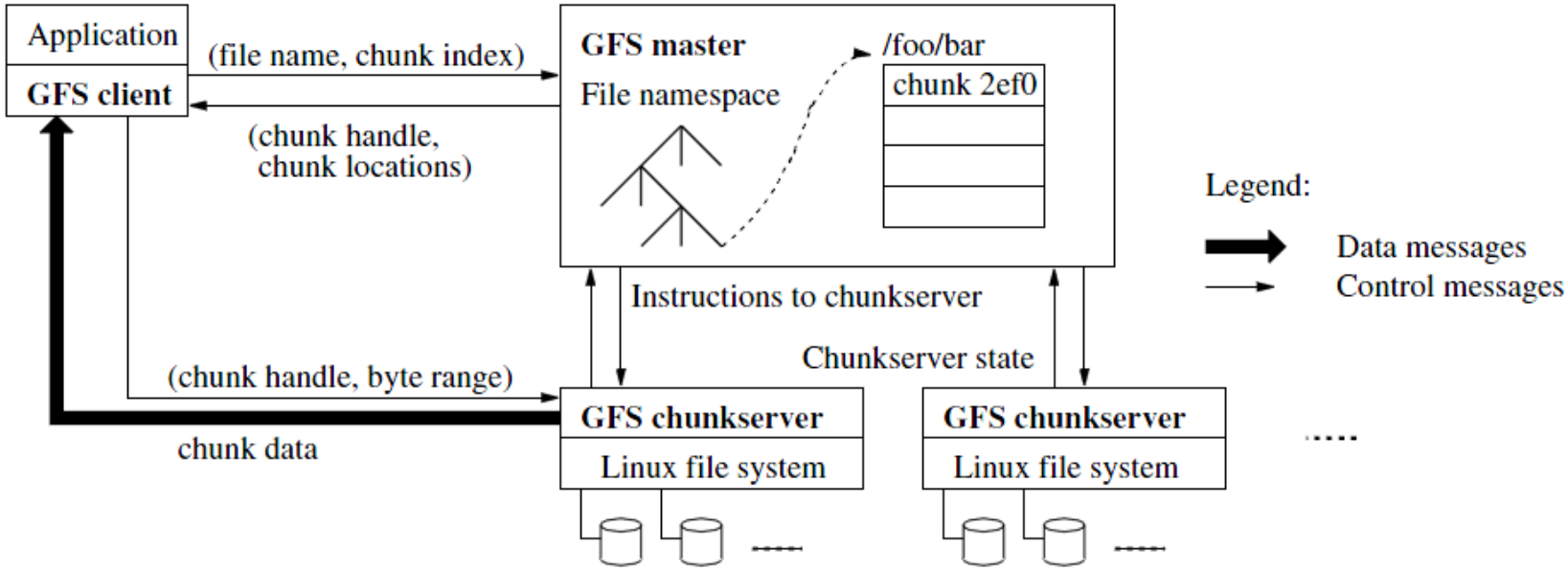
Scalability framework (1)

- scale parameter, or size: k
  - k is carried through into all considered system aspects of interest together
    - system load, e.g., # clients, system capacity, e.g., # servers
- scalability metric, $m(k)$, measure of the system at scale k
  - measure of a quality property (of a system, of an algorithm, ...)
    - e.g. response time, reliability, utilization, number of operations, cost (money)
  - measure of a system resource capacity
    - network diameter, bandwidth between pairs, bisection bandwidth, CPU speed, memory size,
- scalability criterion, $Z(k)$
  - defines a target for $m(k)$
  - expressed in the same units as $m(k)$
  - can be a constant, *e.g.,* a fundamental bound (limit) derivable from other system characteristics independent of the scale parameter

Scalability framework (2)

- scalability is defined as a relation between $m(k)$ and $Z(k)$
  - e.g. $m(k) \geq Z(k)$, $m(k) \sim Z(k)$, $m(k) \leq Z(k)$, $m(k) \rightarrow Z(k)$
    - including a range for which the scaling is considered
- or as an asymptotic growth relation under the ideal assumption that the size can increase indefinitely
  - $m(k) = \mathcal{O}(Z(k)), \Theta(Z(k)), \Omega(Z(k))$
- besides bounds there may be other assumptions that may restrict the validity of the scalability claim
  - e.g. stochastic distributions of system inputs, etc.
  - or assumptions made to simplify the scalability analysis
- often, $Z(k)$ is not made explicit
  - e.g., "system 1 scales better than system 2":
    - $m1(k) \leq m2(k)$ *(i.e.,* $\exists\, Z, K: \forall\, k \in K: m1(k) \leq Z(k) \leq m2(k)$)
  - or: "this system does not scale":
    - the shape of m is (subjectively) discouraging

Example: Google File System



Picture from '**The Google File System',** by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, published at ACM SIGOPS Operating Systems Review - SOSP '03

- Above: model in the deployment view (process view) of GFS
- GFS aims at efficiently and reliably managing many extremely large files for many clients, using commodity hardware (for GFS)
  - hence, many parameters for which scalability questions can be asked

Some questions

This system architecture has been advocated by its developers as highly scalable.

1. Can you imagine why?
2. According to which definition?
3. Do you agree or can you see a problem?

Size is hard to predict:
even for those who cope with it admirably

Kirk McKusick interviewing Sean Quinlan (GFS tech leader)

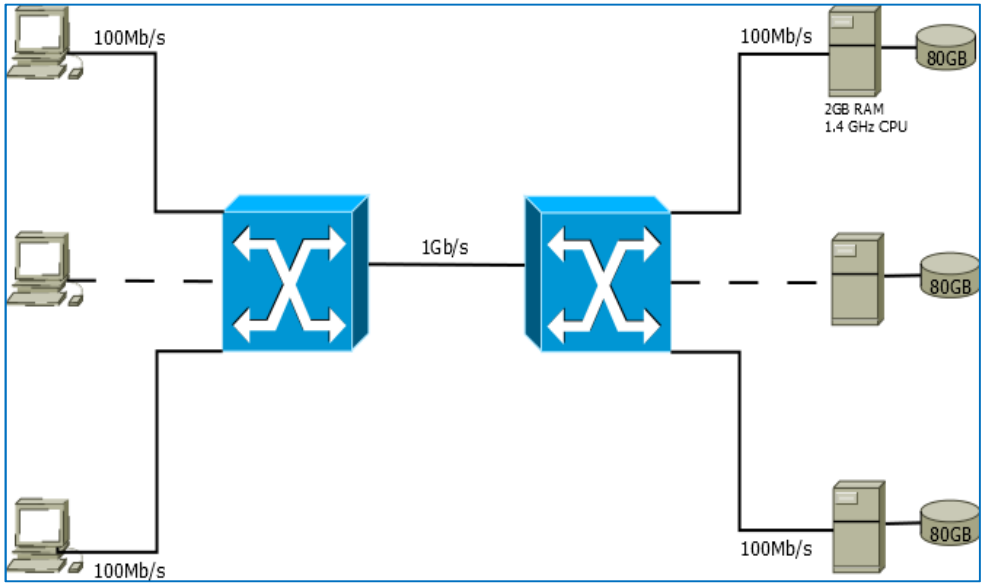
taken from: GFS: Evolution on Fast-forward, ACM QUEUE, Vol.7 Issue 7, August 2009

- QUINLAN … Also, in sketching out the use cases they anticipated, it didn't seem the single-master design would cause much of a problem. The scale they were thinking about back then was framed in terms of hundreds of tera-bytes and a few million files. In fact, the system worked just fine to start with.
- MCKUSICK But then what?
- QUINLAN Problems started to occur once the size of the underlying storage increased. Going from a few hundred terabytes up to petabytes, and then up to tens of petabytes… that really required a proportionate increase in the amount of metadata the master had to maintain. Also, operations such as scanning the metadata to look for recoveries all scaled linearly with the volume of data. So the amount of work required of the master grew substantially.

GFS Measurement

Experimental setting
- 19 servers
  - 1 master + 2 replica's
  - 16 chunk servers
- 16 clients
- Each chunk has 3 replicas

Experiments
1. Each client reads 256 times 4MB randomly selected out of 320GB
2. N clients simultaneously write 1GB to N distinct files
3. N clients append to a single file.

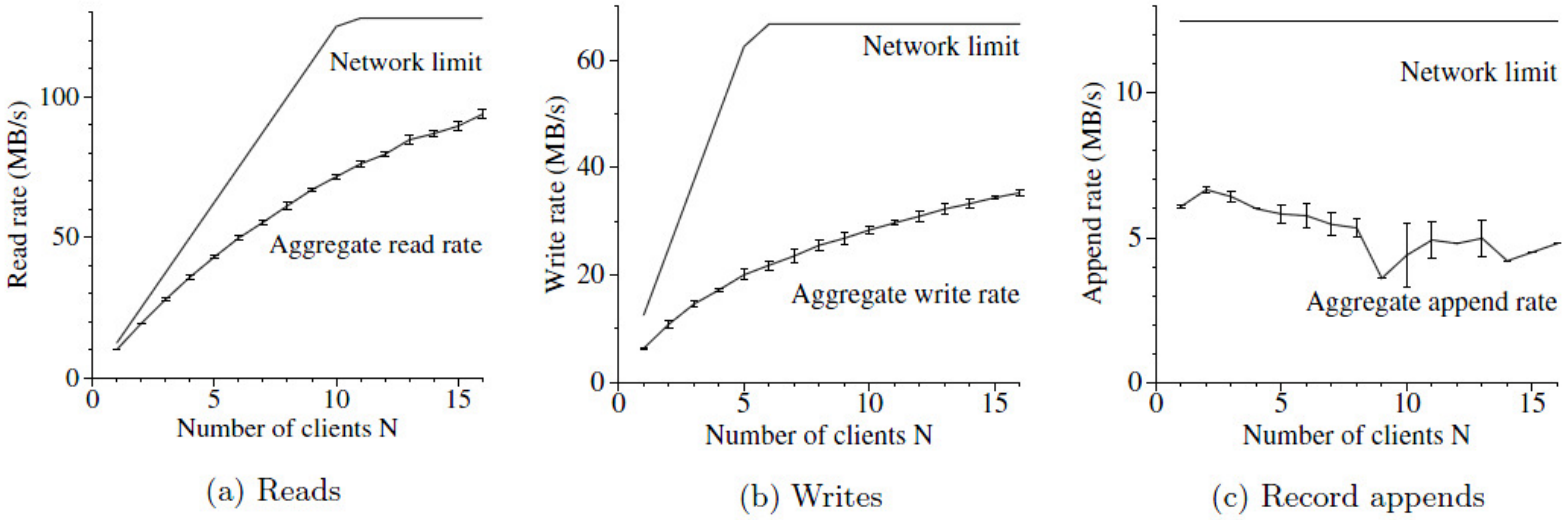


Deployment view of the test setting
- Determines the theoretical limit
  - roofline model
  - sets the target for scalability

Assumption read experiment
- Cache hit rate <= 10% (Why?)

GFS: scaling with number of clients

Picture from '**The Google File System',** by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, published at http://labs.google.com/papers/gfs-sosp2003.pdf



- k: # clients
- $m(k)$: *aggregated* read (write, append) speed, assuming random file access
- $Z(k)$: (not explicitly mentioned): the closer to network limit, the better
- Notes
  - scalability here says something about how efficient resources are used (utilization)
  - explain the shape of the Network limit curve (think of the physical view)
  - what are shapes that indicate bad scalability?
  - what are alternative scalability metrics (mention three)?

Scalability framework (3)

- Scalability is always in terms of a (growth) relation between the scalability metric and the criterion (as a function of the scale parameter k).
  - 'This system is scalable' is a rather pointless expression (or under-specified)
  - always investigate 'what scales with what'

- Normalization: to see the dependence on k, compare to the metric value for a reference value k_0 :
  - examine $m(k)/m(k_0)$ or $m(k_0)/m(k)$
    - depending on behavior, e.g. whether m is increasing or decreasing with k

- Linear scalability: $\frac{m(k)}{m(k_0)} \geq f\frac{k}{k_0}$

  - where f is a positive number
  - dividing by $m(k_0)$ can be regarded as normalization (e.g. $k_0 = 1$)

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 for which it is responsible
- To that end it needs to receive parts of A and B stored elsewhere
- Then, an approximation of the time to execute on P processors is

$$T(P,N) = \frac{2fN^3}{P} + 2cN^2 = \frac{2fN^3}{P}\left(1 + \frac{cP}{fN}\right)$$

  - $2f$ = time for a computation step, being MACs (multiply-accumulates)
  - c = time for communication of an element – simple communication model
  - assumes that entire A and B matrices need to be communicated (ignores local storage)
  - additional delays, including possible communication delays, ignored
  - assumes serialization of computation and communication

Scalability analysis

$T(P,N)$ is a scalability metric. It allows us to examine scaling as function of P and N.

- as function of N it represents the complexity of the parallel algorithm
  - $m(N) = T(P,N)$, follows theoretically the performance of the sequential algorithm
  - though a 'limited-scalable' implementation might limit values of N (see below)

- as function of P, normalization leads to the *speedup*
  - $S(P,N) = T(1,N) \,/\, T(P,N)$
  - $T(1,N)$ has no communication costs

Scalability questions

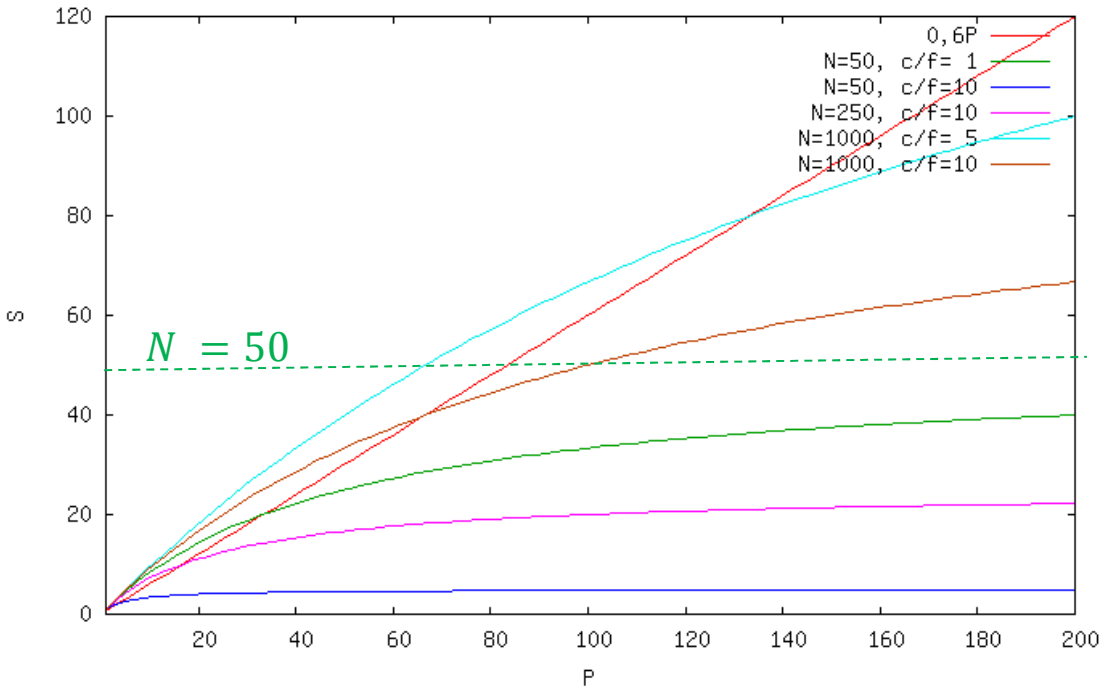
- The speedup can be examined in both dimensions
  - $S(P,N) = (2fN^3) / T(P,N) = P / (1 + (cP/fN))$

- Scalability questions as function of P:
  - e.g. require $S(P,N) \geq Z(P) = 0.6 \cdot P$,
  - this gives a range of P-values for each N

- Scalability questions as function of N:
  - Will the speedup converge to P for large N?
  - How fast does the speedup converge to P (as a function of N)?
  - How large must N be, to have speedup close to P?
  - How many processors can reasonably be used for a size N problem?

Scalability analysis

- Shape of the speedup curve depends on speed of communication relative to computation
  - picture shows slow and faster communication
  - quotient c/f is an important platform parameter for this system



Intersection points

- $S(P,N) = 0.6P$

$P = 0$ or $P = \frac{2fN}{3c}$

$0 \leq P \leq \frac{2fN}{3c}$

$\Leftrightarrow S(P,N) \geq 0.6\ P$

Also we observe

$$\lim_{P \to \infty} S(P,N) = \frac{fN}{c}$$

Amdahl's / Gustafson's law

- Consider a system described with scaling parameters
  - N (problem size, number of jobs or clients per second) and
  - P (system size, number of servers, processors)
- Typically, the performance metric has two parts:
  - a sequential part and a part that can be performed in parallel
    - $m(P,N) = seq(P,N) + par(P,N)$
  - scaling of the system size has no effect on the seq part
    - $seq(P,N) = seq(1,N)$
  - scaling of the system size has linear effect on the par part
    - $par(P,N) = par(1,N)/P$
- e.g., in matrix multiplication
  - $seq(P,N) = 2cN^2$ and $par(P,N) = \frac{2fN^3}{P}$

Amdahl's law

- Keep the problem size N fixed
- Let $\alpha_N = seq(1,N)/par(1,N)$
- Then, we have,
  - $m(1,N) = seq(1,N) + par(1,N) \leq Pseq(1,N) + par(1,N)$
    $= Pseq(P,N) + Ppar(P,N) = P\, m(P,N)$
  - $S(P,N) = \frac{m(1,N)}{m(P,N)} \leq \frac{P\, m(P,N)}{m(P,N)} = P$
    - Not surprising, given the assumption that the par part scales linearly with P
  - $m(P,N) = seq(P,N) + par(P,N) \geq seq(P,N) = seq(1,N)$
  - $S(P,N) = \frac{m(1,N)}{m(P,N)} \leq \frac{m(1,N)}{seq(1,N)} = \frac{seq(1,N) + par(1,N)}{seq(1,N)} = 1 + \frac{1}{\alpha_N}$
    - Bad news, because this gives a fixed (independent of P) upper bound on the speedup. However, notice that the second term may be increasing in N .

Gustafson's law

- Keep $\alpha_{N,P} = \alpha$ fixed $(0 \leq \alpha \ll 1)$
- Increase the problem size N together with the resource size P to maintain α
  - $N_{\alpha,P}$: $seq(P, N_{\alpha,P}) / par(P, N_{\alpha,P}) = \alpha$
- Then, we have
  - $m(1, N_{\alpha,P}) = seq(1, N_{\alpha,P}) + P\, par(P, N_{\alpha,P}) \geq P\, par(P, N_{\alpha,P})$
  - $m(P, N_{\alpha,P}) = seq(P, N_{\alpha,P}) + par(P, N_{\alpha,P}) = (\alpha + 1)\, par(P, N_{\alpha,P})$
  - $S(P, N_{\alpha,P}) \geq \frac{P}{(\alpha+1)} \geq P(1-\alpha)$
    - Good news, because this gives a speedup proportional to the system size
- For the matrix example this means maintaining

$$N_{\alpha,P} = \frac{c}{\alpha f} P$$

General lessons

From Amdahl's law, when solving problems of a fixed size (or dealing with a fixed system load):

- Whenever the cost contribution associated with the variable by which you tune your system has become negligible, move your attention somewhere else (to another system variable).

From Gustafson's law, when system scaling induces overhead costs:

- Ensure that the system load increases in such a way that the cost contribution of overhead stays below a fixed fraction of the total cost.
- If this cannot be done, then probably some more drastic architectural measures are necessary.

A value preserving perspective (1)

Looking only at a single dependency is limited

- implication of Gustafson's law
- typically, a larger system is designed to work with larger problems
- hence, when going to size k, increase a number of relevant, dependent parameters, collectively
- Examples:
  - more servers are installed to address a larger number of clients
  - a larger processor network is use to perform a larger matrix multiplication

A decision must be made

how to jointly change these numbers!

A value preserving perspective (2)

- The value of the system, $v(k)$, is a metric representing the system's "benefit" at scale k
  - e.g. effective #transactions/sec, effective computations/sec
- The cost $C(k)$ of the system at scale k represents a cost measure for creating the system at scale k
  - e.g. # processors including additional ones for increasing connectivity, network or memory resources, or real money
- The metric $v(k)/C(k)$, represents "value for money", a.k.a. "efficiency"
  - Notion of scalable: must be constant, or increasing with k
  - note: $C(k)$ plays the role of $Z(k)$ in our scalability framework

Joint changes

- Scalability question: *if we scale system parameters jointly, do we retain value for the investment?*

- Website
  - increase #servers such that #clients / #servers is roughly constant
- Matrix multiplication
  - increase #processors with factor k
  - how should N change?
    - a factor k would generate a $\sim k$ increase in work capacity
    - hence, increasing N with factor $k^{1/3}$ gives an increase in work load that is comparable to the increase in work capacity

- The metric $v(k)/C(k)$, represents value for money ("efficiency")
  - scalable: must be constant, or increasing with k
  - note: $C(k)$ plays the role of $Z(k)$ in our scalability framework

Matrix multiplication example

- Start situation
  - matrix dimension: N_0 (computational load N_0^3)
  - #processors: *1*
- Scale load and size with factor k*:*
  - matrix dimension: $k^{1/3} N_0$ (to increase load by factor k)
  - #processors*:* k (to increase system size by factor k)
- Value $v(k)$, options:
  - speedup:

    $$v(k) = T(1, k^{1/3}N_0) \,/\, T(k, k^{1/3}N_0) = k \,/\, (1 + ck/(fk^{1/3}N_0))$$
  - effective number of operations per second:

    $$v(k) = kN_0{}^3 \,/\, T(k, k^{1/3}N_0)$$
- Cost, $C(k)$: #processors
- Efficiency $v(k) \,/\, C\,(k)$*:*
  - $\sim 1 \,/\, (1 + ck^{2/3}/(fN_0))$

Matrix multiplication example (ctnd)

- $v(k) \,/\, C\,(k) \;\sim\; 1 \,/\, (\,1 + \, ck^{2/3}/(fN_0)\,)$
  - When k increases, this approaches 0
  - From this perspective, matrix multiplication is not scalable
  - Confirms the analysis from Amdahl's law

- We need to increase the problem size faster in order to have scalability
  - this is because only then the overhead term from the communication is 'overcome'

Exercise:

Show that only if the problem size is scaled linearly with k or more, i.e., $k^e N_0, \;\; e \geq 1$, the value for money metric is constant. Note that this confirms the analysis using Gustafson's law.

Matrix multiplication example (ctnd)

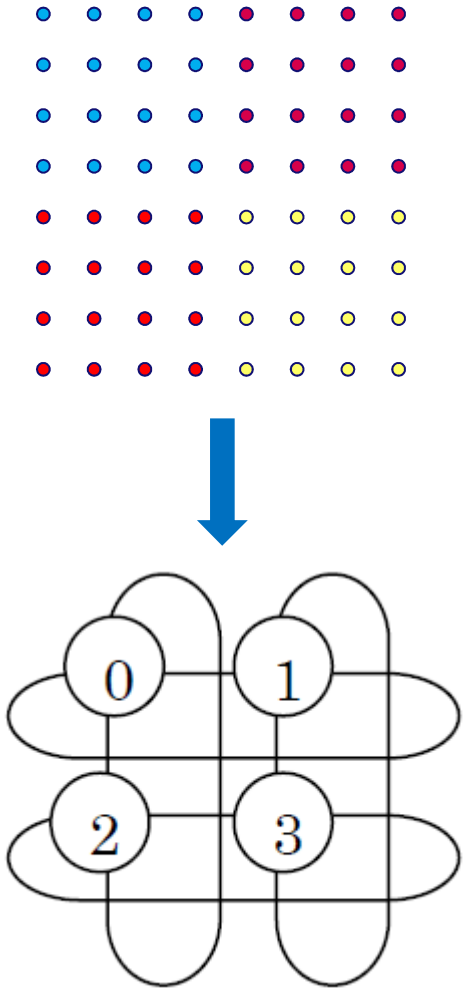
- If the communication would also decrease with the number of processors
  - e.g. $T(P,N) = (2fN^3 + 2cN^2)/P$
  - then $v(k) / C(k) = 1 / (1 + c/(fk^{1/3}N_0))$
    - (goes to 1 for k to infinity)

What to do when scalability is poor?

- Review assumptions to see whether there are opportunities to make architectural changes
  - e.g. from synchronous to asynchronous communication, different scheduling strategies, protocol changes, structural changes, communicate processes instead of data, …
- Assumptions in our example
  - communication time can be seen as an overhead, proportional to the size of the matrices,
    - so can we try to reduce the amount of communication
    - or to do communication in parallel
  - no idle time is incurred, e.g. by waiting on communication resources
    - that has to stay
  - communication and computation don't overlap.
    - that can change, i.e., try to apply latency hiding

Possible realization

- 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
    - assuming hardware support
- Results in better scalability
  - $T(P^2, N) = 2f\frac{N^3}{P^2} + 2c\frac{N^2}{P}$
  - For $P \times P$ PEs



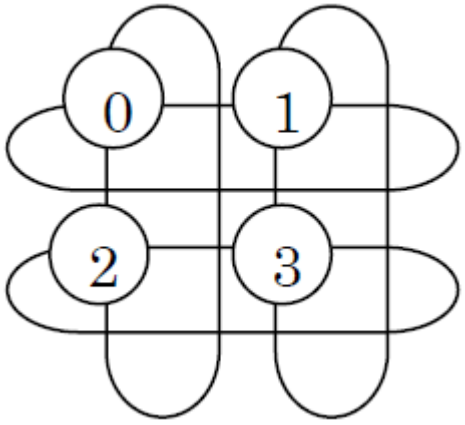
Possible realization (algorithm)

Matrix partition

$$\begin{pmatrix} M_{1,1} & M_{1,2} \\ M_{2,1} & M_{2,2} \end{pmatrix}$$

Algorithm

$$P_0: C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$
$$P_1: C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$
$$P_2: C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$
$$P_3: C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$



Phase 1:

P_0 owns $A_{1,1}$ and $B_{1,1}$
P_1 owns $A_{1,2}$ and $B_{2,2}$
P_2 owns $A_{2,2}$ and $B_{2,1}$
P_3 owns $A_{2,1}$ and $B_{1,2}$

Phase 2:

P_0 owns $A_{1,2}$ and $B_{2,1}$
P_1 owns $A_{1,1}$ and $B_{1,2}$
P_2 owns $A_{2,1}$ and $B_{1,1}$
P_3 owns $A_{2,2}$ and $B_{2,2}$

Non-Scaling

- Suppose communications in the matrix multiplication are done via a shared medium, without local buffering
  - e.g. bus, wireless medium

Exercise: Model $T(P, N)$ again and examine the scalability. Mind the order of local computations.

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?

Beware COST

- **Configuration that Outperforms a Single Thread.**

scalable system	cores	twitter	uk-2007-05
GraphLab	128	249s	833s
GraphX	128	419s	462s
Vertex order (SSD)	1	300s	651s
Vertex order (RAM)	1	275s	-
Hilbert order (SSD)	1	242s	256s
Hilbert order (RAM)	1	110s	-

Table 4: Reported elapsed times for 20 PageRank iterations, compared with measured times for single-threaded implementations from SSD and from RAM. The single-threaded times use identical algorithms, but with different edge orders.

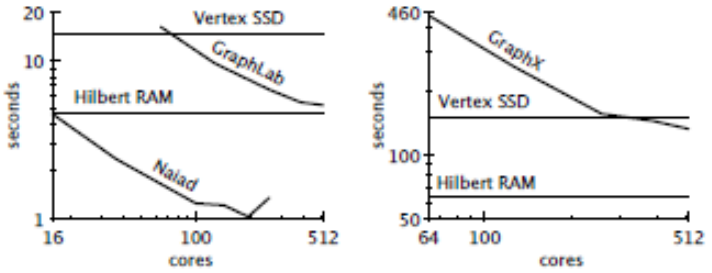


Figure 5: Published scaling measurements for PageRank on twitter_rv. The first plot is the time per warm iteration. The second plot is the time for ten iterations from a cold start. Horizontal lines are single-threaded measurements.

Naiad COST = 16, GraphLab COST = ∞
GraphX Cost > 300

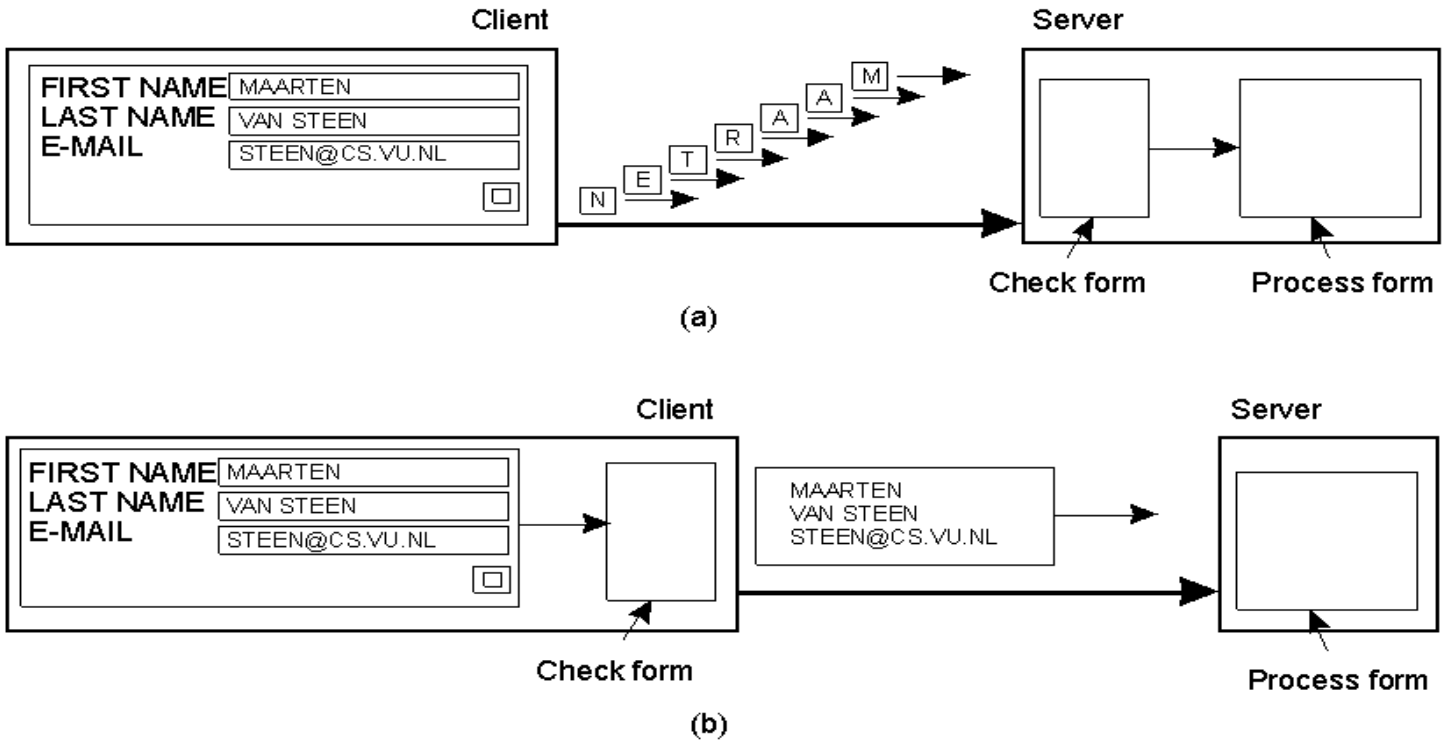
Frank Mc Sherry, Michael Isard, Derek G. Murray
Scalability! But at what COST? HOTOS15
https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry

Scalability tactics

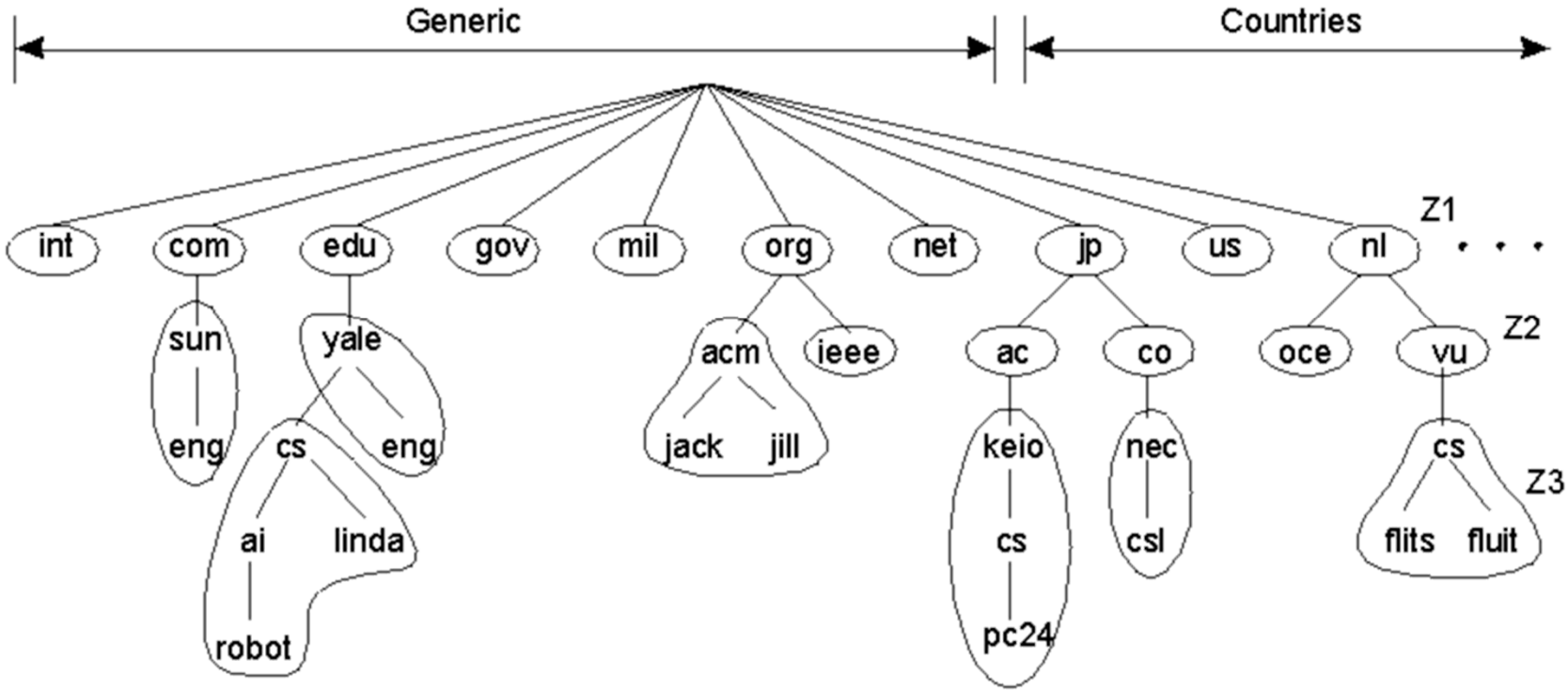
- Rules of thumb (towards scalability): limit dependencies
  - No machine needs/has complete information
  - Decisions are based on local information
  - Failure of one machine does not ruin results
  - No assumption of a global, shared clock

- 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 neighbouring tasks

Scaling Techniques (Reduce overall communication)



Scaling Techniques (partition and distribute)



- Dividing the DNS space into zones benefits from
  - locality principle
  - task (resolution) and data (DNS database) distribution
  - information replication (caching)

Extra-functional properties and architecture

- 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
- This question also relates to more fine-grained choices to be made
  - e.g. the nature of the connectors (message passing, RPC, ...)
- Main views addressed in the following presentations:
  - process and deployment views,
  - and logical organization of the development view

Literature

- Charles B. Weinstock, John B. Goodenough, *On System Scalability* , CMU/SEI-2006-TN-012, 2006
- 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.
- A.B. Bondi, *Characteristics of Scalability and Their Impact on Performance*, in *Proc. 2nd WOSP,* ACM Press, 2000, pp 195-203.
- Ananth Grama, Anshul Gupta, GeorgeKarypis and Vipin Kumar, *Introduction to parallel computing,* 2nd edition, Addison Wesley, 2003, Chapter 5.
- Ian Foster, *Designing and Building Parallel Programs*, Addison Wesley,1995, Chapter 3.
  - (available on the Web, http://www.mcs.anl.gov/dbpp)