Scalability in Distributed Systems

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

Types of Scalability (Bondi 2000)

- A system has *Load Scalability*
  - If it has the ability to function gracefully i.e., without undue delay or unproductive resource consumption and contention over a range of system loads. E.g., WiFi/Ethernet does not have load scalability.
- A system has *Space Scalability*
  - If its memory requirements do not grow to intolerable levels as the number of items supported increases. E.g., a data structure is space scalable w.r.t. to the number of items it represents if its memory requirements increase sublinearly w.r.t # of objects.
- 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*

Need for a scalability framework

Prior definitions are interesting but not “good enough”, because they are

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.

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
  - e.g. # clients, # servers, load, input size of an algorithm etc.
- 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 f*or* $m(k)$
  - 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. $m(k) \leq Z(k)$, $m(k) \sim Z(k)$ ...
    - 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
  - $m(k) = \Omega(Z(k))$, $m(k) = \Theta(Z(k))$
- 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

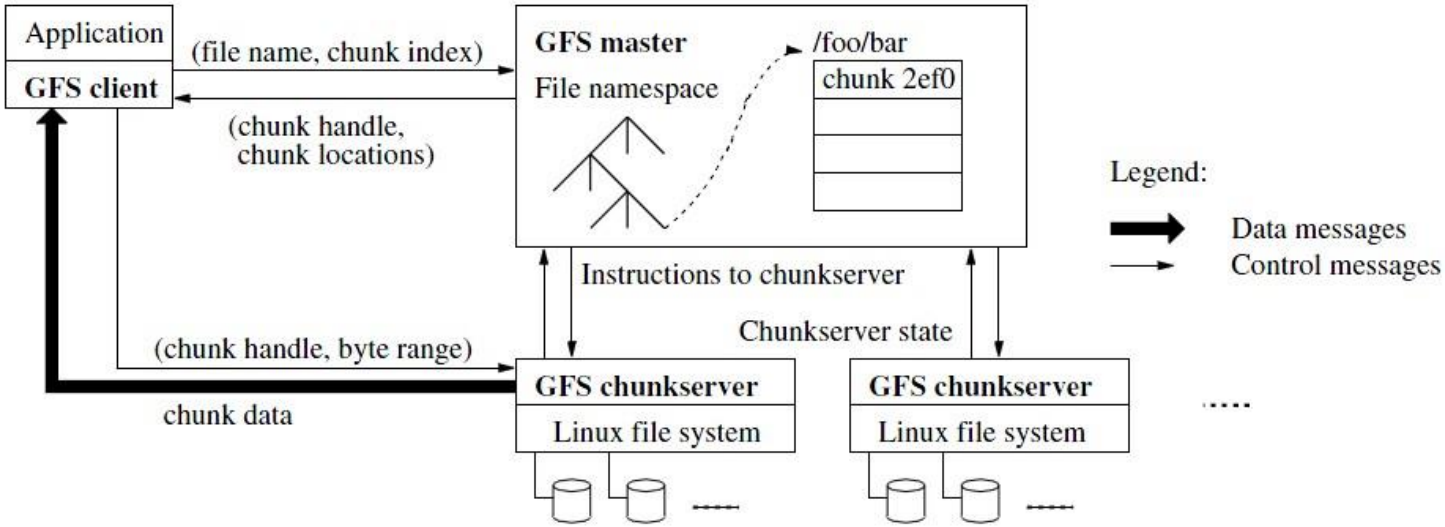
Scalability framework (3)

- 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 underspecified)
  - always investigate 'what scales with what'

- reference: compare with $k = k_0$ as reference to see the dependence on k:
  - 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: $\frac{m(k)}{m(k_0)} \leq f\frac{k}{k_0}$
  - where f is a positive number
  - dividing by $m(k_0)$ can be regarded as normalization (e.g. $k_0 = 1$)

Example: 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)

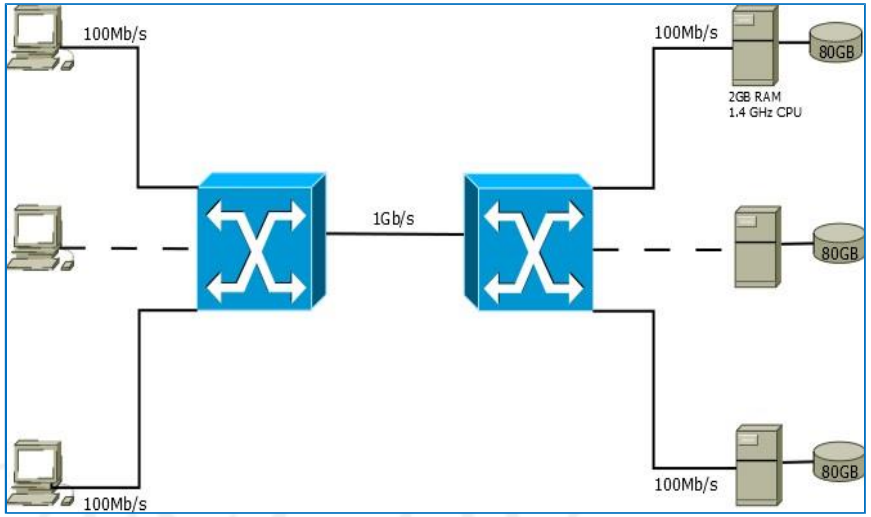
– hence, many parameters for which scalability questions can be asked

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.

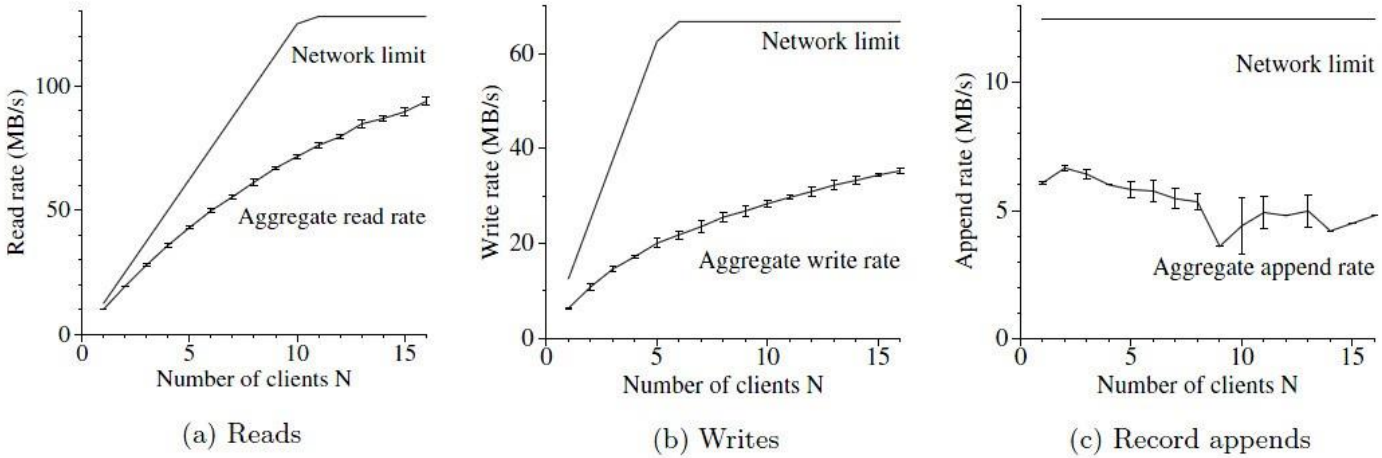
Assumption read experiment
- Cache hit rate <= 10% (Why?)



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

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?

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 (recall sublinear increase in representation) with the volume of data. So the amount of work required of the master grew substantially.

Architecture scalability

- Different ways to scale a system
  - Vertical scalability (scale up) by adding resources to the single node / improve existing code to work better
  - Horizontal scalability (scale out) by adding more nodes to the system
- Scalability - vertical
  - Add: CPU, Memory, Disks (bigger box)
  - Handling more simultaneous
    - Connections, operations, users
- Choose a good I/O and concurrency model
  - Non blocking I/O, Asynchronous I/O, Threads (single, pool, per-connection)
- Scalability – horizontal
  - Add more machines/sfotware cooperating toward a goal