Rules of Thumb in Data Engineering

Jim Gray, Prashant Shenoy

December 1999

Revised March 2000

Technical Report
MS-TR-99-100

Microsoft Research
Advanced Technology Division
Microsoft Corporation
One Microsoft Way

Redmond, WA. 98052

Rules of Thumb in Data Engineering

Jim Gray, Prashant Shenoy
Microsoft Research, U. Mass, Amherst.
Gray@Microsoft.com, shenoy@cs.umass.edu

Abstract

This paper reexamines the rules of thumb for the design of data storage systems. Briefly, it looks at storage, processing, and networking costs, ratios, and trends with a particular focus on performance and price/performance. Amdahl's ratio laws for system design need only slight revision after 35 years—the major change being the increased use of RAM. An analysis also indicates storage should be used to cache both database and web data to save disk bandwidth, network bandwidth, and people's time. Surprisingly, the 5-minute rule for disk caching becomes a cache-everything rule for web caching.

1. Introduction

We engineer data using intuition and rules of thumb. Many of these rules are folklore. Given the rapid changes in technology, these rules need to be constantly re-evaluated.

This article is our attempt to document some of the main rules we use in engineering database systems. Since we have to design for the future, the article also assesses technology trends and predicts the sizes of future systems.

2. Storage performance and price

Many rules of thumb are a consequence of *Moore's Law*, which posits that circuit densities increase four fold every three years. That means that memories get four times larger each three years, or about 100x per decade. It also means that in-memory data grows at this rate: creating the need for *an extra bit of addressing every 18 months*. In 1970 we were comfortable with 16-bit address spaces: it was rare to find a machine with a mega-word of memory. Thirty years later we need 20 extra address bits to address the 64 GB memories (36 bit addresses) found in the larger computers on the market. Today most computer architectures give 64-bit logical addressing (e.g. MIPS, Alpha, PowerPC, SPARC, Itanium) or 96-bit (e.g. AS400) addressing. Physical addressing is 36-bits to 40-bits, and growing a bit per 18 months. At this rate it will take two or three decades to exceed 64-bit addressing.

Moore's Law originally applied only to random access memory (RAM). It has been generalized to apply to microprocessors and to disk storage capacity. Indeed, disk capacity has been improving by leaps and bounds; it has improved 100 fold over the last decade. The magnetic aerial density has gone from 20 Mbpsi (megabits per square inch in 1985), to 35 Gbpsi late in 1999. Disks spin three times faster now, but they are also 5 times smaller than they were 15 years ago, so the data rate has improved only 30 fold (see Figure 1). Today, disks can store over 70 GB, have access times of about 10 milliseconds (~ 120 (Kaps kilobyte accesses per second)), and transfer rates of about 25MBps (~ 20 Maps (megabyte accesses per second)) and a scan time of 45 minutes [1]. These disks cost approximately 42 k$/TB today (15 k$/TB for lower-performance IDE drives packaged, powered, and network served) [2]. Within 5 years, the same form-factor should be storing nearly ½ terabyte, support 150 Kaps, and have a transfer rate of 75 MBps. At that rate, it will take nearly 2 hours to scan the disk. By then, the prices should be nearing 1 k$/TB (including server).

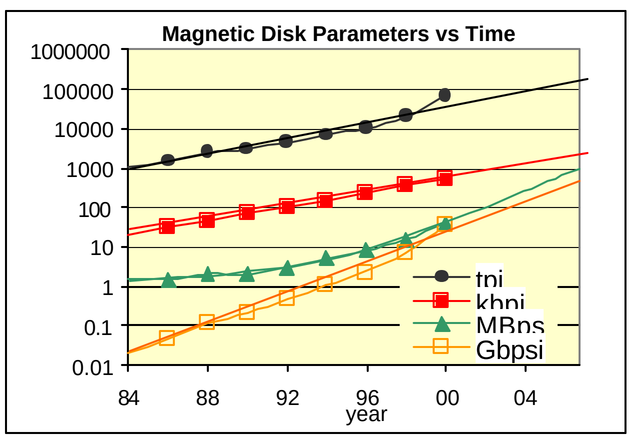

Figure 1: Disk capacity has improved 1,000 fold in the last 15 years, consistent with Moore's law, but the transfer rate MBps has improved only 40x in the same time. The metrics are tracks per inch (tpi), thousands of bits per linear inch of track (kbpi), mega bytes per second as the media spins (MBps), and gigabits per square inch (Gbpsi).

This paper appeared in the proceedings of the IEEE International Conference on Data Engineering, Feb 28-30, San Diego, CA.

The ratio between disk capacity and disk accesses per second is increasing more than 10x per decade. Also, the capacity/bandwidth ratio is increasing by 10x per decade. These changes have two implications: (1) disk accesses become more precious; and (2) disk data must become cooler (have fewer accesses per byte stored) [3].

We reduce disk accesses by (1) using a few large transfers rather than many small ones, (2) favoring sequential transfers, and (3) using mirroring rather than RAID5. To elaborate on these three points. (1) We can reduce disk accesses by caching popular (hot) pages in main memory, and writing a log of their changes to disk. This reduces random reads, and converts random writes to sequential (log) writes. Periodically, the written data needs to be checkpointed to disk to minimize redo work at restart, but often this checkpoint work can be done in the background piggybacking on other IOs, and can be sorted so that it is nearly sequential. These important optimizations are used by database systems today. Over the last decade, disk pages have grown from 2KB to 8KB and are poised to grow again. In ten years, the typical *small* transfer unit will probably be 64KB, and large transfer units will be a megabyte or more. (2) A random access costs a seek time, half a rotation time, and then the transfer time. If the transfer is sequential, there is no seek time, and if the transfer is an entire track, there is no rotation time. So track-sized sequential transfers maximize disk bandwidth and arm utilization. The move to sequential disk IO is well underway. As already mentioned, caching, transaction logging, and log-structured file systems convert random writes into sequential writes. This has already had large benefits for database systems and operating systems. These techniques will continue to yield benefits as disk accesses become even more precious. (3) Both RAID5 (parity) and RAID1 (mirrors) offer fault-tolerant disk storage. Since IOs are the scarce resource, one wants to optimize for IOs rather than for space. The argument in favor of mirrors versus RAID5 is that mirrored disks offer double the read bandwidth to each data item, and they cost only one extra access for a write. RAID5 uses up to four disk accesses to do a write, and improves read bandwidth only if the data requests go to different disks. RAID5 saves disk space (gigabytes) at the expense of more IOs for disk writes.

Ten years ago, disks offered 50 Kaps (kilobyte accesses per second) to 1GB of data, and 5-minute disk scan times. Current disks offer 120 Kaps to 80 GB of data with a 45-minute scan times. This is 1 Kaps per 20MB in 1990 vs. 1 Kaps per 500MB now. So, modern disk data needs to be at 25x colder than data of 10 years ago. In fact, all the “hot” data of 1990 has migrated to RAM: disk cost 10$/MB in that era, five times what RAM costs today. So 1990s disk data can afford to live in RAM today. The use of large main memories is one way to cool the data on disk. Another way is to store the data multiple times and spread the reads among the copies: again suggesting mirroring.

Meanwhile, there has been great progress in tape storage: tapes now store 40 GB. A drive with a 15 tape cartridges costs about 10k$ and stores about 600GB nearline. These drives provide 6 MBps data rates, so the scan time for all the cartridges is about 1.2 days. Such nearline tape archives deliver approximately zero Kaps and Maps (10^{-2} Kaps is typical). Such a tape archive is half the cost per terabyte of disk storage, but tape does not provide easy access to the data. The cost per random tape access is about a hundred thousand times higher (100 accsses/second/1K$ disk versus 000.01 accesses/second/10,000$ tape). In five years, this situation should be even more dramatic -- a million-to-one is compelling. Tape capacities are expected to improve faster than tape speed, and access time is expected to stay about the same, making the access problem even more problematic: several days to scan a tape archive. This suggests nearline-tape will be purely archival.

Historically, tape, disk, and RAM have maintained price ratios of about 1:10:1000. That is, disk storage has been 10x more expensive than tape, and RAM has been 100x more expensive than disk. Indeed, today one can buy a 40 GB tape cartridge for 80$, a 36 GB disk for 1200$, and 1 GB of memory for about 2400$ [4] (DELL and SCSI are not the least expensive). These ratios translate to 2$/GB, 32 $/GB and 2.4k$/GB giving a ratio of 1:16:1200 for storage.

But, when the offline tapes are put in a nearline tape robot, the price per tape rises to 10K$/TB while packaged disks are 30K$/TB. This brings the ratios back to 1:3:240. It is fair to say that the storage cost ratios are now about 1:3:300.

The cost/MB of RAM declines with time: about 100x per decade. Since disk and RAM have a 1:100 price ratio, this price decline suggests that *what is economical to put on disk today will be economical to put in RAM in about 10 years*.

A striking thing about these storage cost calculations is that disk prices are approaching nearline tape prices. By using RAID (mirroring or parity), administrators sacrifice disk storage capacity to protect against disk media failures. Administrators are discovering that you may be able to backup a terabyte to tape, but it takes a very long time to restore a terabyte from tape. As they see petabyte stores looming on the horizon, administrators are moving to strategies that maintain multiple disk versions online so that one never has to restore the database from tape. Increasingly, sites that need to be online all the time are replicating their entire state at a remote site, so that they have two online copies of the data. If one site fails, the other offers access to the data, and the failed site can recover from the data stored at the second site. In essence, *disks are replacing tapes as backup devices.* Tapes continue to be used for data interchange, but if Gilders' Law holds (see below), then someday all data interchange will go over the Internet rather than over sneaker net. That means tape will be less frequently used for data interchange.

Storage prices have dropped so low that the storage management costs now exceed storage hardware costs (similarly, PC management costs exceed the cost of the hardware). In 1980, there was a rule of thumb that one needed a data administrator for 1GB of storage. At that time a GB of disk cost about a million dollars, and so it made sense to have someone optimizing it and monitoring the use of disk space. Today, a million dollars can buy 1 TB to 100 TB of disk storage (if you shop carefully). So, today, the rule of thumb is that a person can manage 1 TB to 100 TB of storage – with 10 TB being typical. The storage management tools are struggling to keep up with the relentless growth of storage. If you are designing for the next decade, you need build systems that allow one person to manage a 10 PB store.

Summarizing the Storage rules of thumb:

1. *Moore's Law: Things get 4x better every three years.*
2. *You need an extra bit of addressing every 18 months.*
3. *Storage capacities increase 100x per decade.*
4. *Storage device throughput increases 10x per decade.*
5. *Disk data cools 10x per decade.*
6. *Disk page sizes increase 5x per decade.*
7. *NearlineTape:OnlineDisk:RAM storage cost ratios are approximately 1:3:300.*
8. *In ten years RAM will cost what disk costs today.*
9. *A person can administer a million dollars of disk storage: that is 30TB of storage today.*

And two observations:

* *Disks are replacing tapes as backup devices.*
* *On random workloads, disk mirroring is preferable to RAID5 parity because it spends disk space (which is plentiful) to save disk accesses (which are precious).*

3. Amdahl's system balance rules

Gene Amdahl is famous for many rules of thumb. For data engineering, there are four famous ones [6]:

10. *Amdahl's parallelism law: If a computation has a serial component S and a parallel component P, then the maximum speedup is (S+P)/S.*
11. *Amdahl's balanced system law: A system needs a bit of IO per second for each instruction per second: about 8 MIPS per MBps.*
12. *Amdahl's memory law: ? ? 1: that is, in a balanced system the MB/MIPS ratio, called alpha (?), is 1.*
13. *Amdahl's IO law: Programs do one IO per 50,000 instructions.*

How have Amdahl's laws changed in the last 35 years? The parallelism law is algebra, and so remains true and very relevant to this day. The thing that is surprising is that the other 35-year-old "laws" have survived while speeds and sizes have grown by orders of magnitude and while ratios have changed by factors of 10 and 100.

To re-evaluate Amdahl's IO laws, one can look at the Transaction Processing Performance Council benchmark systems [4]. These systems are carefully tuned to have the appropriate hardware for the benchmark. For example, the OLTP systems tend to use small disks because the benchmarks are arm limited, and they tend to use the appropriate number of controllers. The following paragraphs evaluate Amdahl's balanced system law: concluding that with current technology it should be amended to say:

10. *Amdahl's revised balanced system law: A system needs 8 MIPS/MBpsIO, but the instruction rate and IO rate must be measured on the relevant workload. (Sequential workloads tend to have low CPI (clocks per instruction), while random workloads tend to have higher CPI.)*
12. *Alpha (the MB/MIPS ratio) is rising from 1 to 4. This trend will likely continue.*
13. *Random IO's happen about once each 50,000 instructions. Based on rule 10, sequential IOs are much larger and so the instructions per IO are much higher for sequential workloads.*

Amdahl's balanced system law becomes more complex to interpret in the new world of quad-issue pipelined processors. Table 2 summarizes the following analysis. In theory, the current 550 MHz Intel processors are able to execute 2 billion instructions per second, so Amdahl's IO law suggests that each 550 MHz processor needs 160 MBps of disk bandwidth (all numbers rounded). However, on real benchmarks, these processors demonstrate 1.2 clocks per instruction (CPI) on sequential workloads (TPC-D,H,R) and 2.2 clocks per instruction on random IO workloads (TPC-C, W) [7,8]. These larger CPIs translate to 450 MIPS on sequential and 260 MIPS on random workloads. In turn, Amdahl's law says these processors need 60 MBps sequential IO bandwidth (~450/8) and 30 MBps random of IO bandwidth (~260/8) per cpu respectively (for tpcH and tpcC). A recent tpcH benchmark by HP [5] used eight 550 MHz processors with 176 disks. This translates to 22 disks per cpu, or about 70 MBps of raw disk bandwidth per cpu and 120 MBps of controller bandwidth per cpu (consistent with Amdahl's prediction of 60MBps). Amdahl's law predicts that system needs

30MBps of IO bandwidth. Using 8KB pages and 100 IO/s per disk implies 38 disks per processor – a number comparable to the 50 disks Dell actually used [4].

Both TPC results mentioned here use approximately ½ gigabyte of RAM per processor. Based on the MIPS column of Table 2, the TPC systems have approximately 1 to 2 MB per MIPS. These are Intel IA32 processors that are limited to 4 GB of memory. When one considers HP, IBM, and Sun systems that do not have the 4GB limit, there is between 1GB/cpu and 2.5GB/cpu (12 to 64 GB overall). This roughly translates to a range of between 2 MB/MIPS 6 MB/MIPS. As argued by many main memory database advocates (e.g. [9]), as disk IOs become more precious, we are moving towards relatively larger main memories. Alpha, is rising from 1 to 4.

What about the execution interval? How many instructions are executed per IO? In essence, if 8 instructions are executed per byte of IO (law 10), and if 50 K instructions are executed per IO (law 13), then IOs are about 6 KB (~50/8). Again, there is a dichotomy between sequential and random workloads: On TPC-C benchmarks which do a lot of random IO, there are about 60 k instructions between 8 KB IOs (~7*8) and on TPC-H sequential workloads there are 200 k instructions between 64 KB IOs (~3*64).

In summary, Amdahl's laws are still good rules-of-thumb in sizing the IO and memory systems. The major changes are that (1) the MIPS rate must be measured, rather than assuming a CPI of 1 or less, (2) sequential IOs are much larger than random IOs and hence the instructions per IO are much higher for sequential workloads, (3) Alpha (the MB/MIPS ratio) is rising from 1 to 2 or 4. This trend will likely continue. Given the 100x and 1,000x changes in device speeds and capacities, it is striking that Amdahl's ratios continue to hold.

Interestingly, Hsu, Smith, and Young, came to similar conclusions in their very detailed study of TPC-C and other workload behaviors [10]. Their excellent study shows the wide spectrum of behaviors, both across workloads, and within a given workload.

Table 2: Amdahl's balanced system law and the parameters of two recent TPC benchmarks (www.tpc.org). The CPI varies among the workloads, and the IO sizes also vary, still, the instructions/byte are similar to Amdahl's prediction of eight instructions per byte (a bit of IO per instruction).

	MHz/ cpu	CPI	mips	KB/ IO	IO/s/ disk	Disks	Disks/ cpu	MB/s/ cpu	**Ins/ IO Byte**
Amdahl	1	1	1	6					**8**
TPC-C = random	550	2.1	262	8	100	397	50	40	**7**
TPC-H = sequential	550	1.2	458	64	100	176	22	141	**3**

4. Networking: Gilder's Law

George Gilder predicted in 1995 that network bandwidth would triple every year for the next 25 years [13]. So far his prediction seems to be approximately correct. Individual fiber optic wavelength channels run at 40 Gbps. Wave-division multiplexing gives 10 or 20 channels per fiber. Multi-terabit links are operating in the laboratory on a single fiber. Several companies are deploying thousands of miles of fiber optic networks. We are on the verge of having very high-speed (Gbps) wide-area networks. When telecom deregulation and the subsequent competition takes hold, these links will be very inexpensive.

14. *Gilder's law: Deployed bandwidth triples every year.*
15. *Link bandwidth improves 4x every 3 years.*

Paradoxically, the fastest link on the Microsoft campus is the 2.5 Gbps WAN link to the Pacific Northwest GigaPOP. This inverts the speed ratios between WANS and LANs. It takes three 1 Gbps Ethernet links to saturate the WAN link. LAN speeds are about to rise to 1 Gbps, and then to 10 Gbps via switched point-to-point networking.

Latency due to the speed of light will be with us forever -- 60 ms round trip within North America, within Europe, and within Asia. However, terabit-per-second bandwidth will allow us to design systems that cache data locally, and quickly access remote data if needed.

The cost of sending a message is [11]:

Time = senderCPU + receiverCPU + bytes/bandwidth

Traditionally, high-speed networking has been limited by software overheads. The sender and receiver cpu costs have typically been 10,000 instructions and then 10 instructions per byte. So to send 10 KB cost 120,000 instructions or something like a millisecond of cpu time. The transmit time of 10,000 bytes on 100 Mbps Ethernet is less than a millisecond – so the LAN was cpu limited, not transmit time limited.

A rule of thumb for traditional message systems has been

16. *A network message costs*
 10,000 instructions and 10 instructions per byte.
17. *A disk IO costs*
 5,000 instructions and 0.1 instructions per byte.

Why are disk IOs so efficient when compared to network IO? After all, disk IOs are just messages to the disk controller – a storage network message rather than a LAN or WAN message. There have been substantial strides in understanding that simple question. The networking community has offloaded much of the tcp/ip protocol to the NICs (much as SCISI and IDE/ATA do), and the networking software now uses memory more aggressively to buffer requests and correct errors. Checksumming, fragmentation/assembly, and DMA have all been added to high-speed NICs. Much of this work has gone on under the banner of System Area Networking (SAN) and the Virtual Interface Architecture [12]. The current revision to rule of thumb is:

18. The cpu cost of a SAN network message is 3,000 clocks and 1 clock per byte.

It is now possible to do an RPC in less than 10 microseconds, and to move a Gbps from node to node while the processor is only half busy doing network (tcp/ip) tasks. The network carries 100,000 packets per second (300 M clocks according to rule 18) and 128 M bytes per second (128 M clocks according to rule 18) so a 650 MHz machine has 200 M clocks to spare for useful work.

Currently, it costs a more than a dollar to send 100MB via a WAN (see Table 7 of Odlyzko [14]), while local disk and LAN access are 10,000 times less expensive. This price gap is likely to decline to 10:1 or even 3:1 over the next decade. As suggested in subsequent sections, when bandwidth is sufficient and inexpensive, local disks will act as caches for commonly used data and a buffer for pre-fetched data.

5. Caching: Location, Location, and Location

Processor clock speeds have been improving, as has the parallelism within the processor. Modern processors are capable of issuing four or more instructions in parallel and pipelining instruction execution.

In theory, current quad-issue Intel processors are able to execute two billion instructions per second -- 4 instructions per clock and 550 M clocks per second. In practice, real benchmarks see CPI (clocks per instruction) of 1 to 3. The CPI is rising as processor speeds outpace memory latency improvements [6,7,8].

The memory subsystem cannot feed data to the processor fast enough to keep the pipelines full. Architects have added 2-level and 3-level caches to the processors in order to improve this situation, but if programs do not have good data locality, there is not much the architects can do to mask "compulsory" cache misses.

Software designers are learning that careful program and data placement and cache sensitive algorithms with good locality give 3x speedups on current processors. As processor speeds continue to outpace memory speeds, there will be increasing incentives for software designers to look for algorithms with small instruction cache footprints, with predictable branching behavior, and with good or predictable data locality (i.e., clustered or sequential access).

There is a hardware trend to design huge (256 way) multiprocessors that operate on a shared memory. These systems are especially prone to *instruction stretch* in which bus and cache interference from other processors causes each processor to slow down. Getting good performance from these massive SMPs will require careful attention to data partitioning, data locality, and processor affinity.

An alternative design opts for many nodes each with its own IO and bus bandwidth and all using a dataflow programming model and communicating via a high-speed network [15]. These designs have given rise to very impressive performance, for example, the sort speed of computer systems has been doubling each year for the last 15 years through a combination of increased node speed (about 60%/year) and parallelism (about 40%/year). The 1999 terabyte sort used nearly 2,000 processors and disks, http://research.microsoft.com/~gray/sort_benchmark.

The argument for the many-little scalable design tries to leverage the fact that mainframe:mini:commodity price ratios are approximate 100:10:1. That is, mainframes cost about 100 times more than commodity components, and semi-custom mini-computers have a 10:1 markup over commodity components (see prices for comparable systems at the www.tpc.org benchmarks). The cluster advocates admit the many-little design is less efficient, but they argue that it is more cost-effective.

There seems no good general rule of thumb for cpu-caches beyond bigger-is-better and locality-is-better. But, two good rules have evolved for disk data locality and caching. It is possible to quantitatively estimate when you should cache a disk page in memory: trading off memory consumption against disk arm utilization.

As mentioned before, disk arms are precious. If a disk costs $1200 and does 120 accesses per second, then a disk access per second costs $10. It would be advantageous to spend up to $10, to save one access per second. Well, $10 buys about 10MB of RAM, so if a cache of that size would indeed save one access per second, it would be a good investment.

This suggests the question: How frequently must a disk-resident object be accessed to justify caching it in main memory?" When does the rent of RAM space balance the cost of an access? The analysis in [16] shows that:

$$\text{BreakEvenReferenceInterval (seconds)} = \frac{\text{PagesPerMBofRAM}}{\text{AccessPerSecondPerDisk}} \times \frac{\text{PricePerDiskDrive}}{\text{PricePerMBofDRAM}}$$

For randomly accessed data, the first term (call the *technology ratio*) is approximately 1; the second term (called the *economic ratio*) varies from 100 to 400 today.

So, the breakeven interval is about 2 minutes to 5 minutes for randomly accessed pages.

For sequentially accessed data the technology ratio is approximately 0.1 (1MB "pages" and 10 pages per second) so the break-even interval is 10 to 40 seconds.

This analysis gives the rules:

19. *The 5-minute random rule: cache randomly accessed disk pages that are re-used every 5 minutes.*
20. *The 1-minute sequential rule: cache sequentially accessed disk pages that are re-used within a minute.*

Both of these time constants are rising slowly as technology evolves.

A related rule that has not seen much use is that one can *spend 1 byte of RAM to save 1 MIPS*. The argument goes that RAM costs about 1$/MB and today one can get 100 extra MIPS from Intel for 100 extra dollars (approximately). So, the marginal cost of an instruction per second is approximately the marginal cost of a byte. Fifteen years ago, the ratio was 10:1, but since then Intel and VLSI has made processors much less expensive.

21. *Spend 1 byte of RAM to save 1 instruction per second.*

Now consider web page caching. Logic similar to the five-minute rule suggests when it pays to cache web pages. The basic diagram is shown in Figure 2, where the link speed varies from 100 KBps for intranets, to modem speeds of 5 KBps, to wireless speeds of 1 KBps. In case of a modem and wireless links, assume a local browser cache. For high-speed links, the cache could either be a browser cache or a proxy cache. In case of a proxy, assume a fast connection between the user and the cache (e.g., a 100Mb/s LAN), so that the time cost of accessing data from a remote proxy disk is not significantly larger than that from a local disk.

Given these assumptions consider three questions:
(1) How much does web caching improve response times?
(2) When should a web page be cached?
(3) How large should a web cache be?

Assume that the average web object is 10KB. Define
R_remote: response time to access an object at server.
R_local: response time to access the object fromcache.
H: cache hit ratio (fraction of requests cache satisfies).

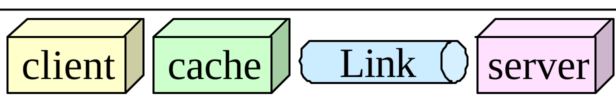

Figure 2. The client-side or proxy web cache improves response time by eliminate link transmission times and server times.

Then: *Response_Time_Improvement* =
*R_remote - (H * R_local + (1-H) * R_remote)* =
*H * (R_remote - R_local)*

R_remote consists the server response time and the download network time. The server response time (the queuing delay and the service time) can range from several hundred milliseconds to several seconds. Assume a response time of 3 seconds.

The download time over the network depends on network conditions and on link speeds. WAN Links are typically shared, so the user bandwidth is smaller than the typical link bandwidth (a bottlenecked link at the server may further reduce the bandwidth/request). Assume that the effective LAN/WAN bandwidth is 100KB/s; hence time to transmit a 10KB object is a tenth of a second. With these assumptions, the *R_remote* is dominated by the 3 second server response time.

Modem bandwidth available on a dial-up link is 56 Kbs. With compression, the effective bandwidth is often twice that, but there are also start/stop overheads. Assume an effective modem bandwidth of 5KB/s. Hence, the modem transmit time for a 10 KB object is 2 seconds, and *R_remote* is 5 seconds.

A mobile user on a wireless link gets 1KB/s, and so it takes 10 seconds to download a 10KB object and *R_remote* is 13 seconds. This is why servers for mobile systems often compress the data to make the objects much smaller (1KB rather than 10KB). Summarizing, *R_remote* can be estimated as:
R_remote
= 3 + .1 = 3s (high speed connection)
= 3 + 2 = 5s (modem connection)
= 3 + 10 = 13s (wireless connection)

R_local depends many details, but fundamentally local access avoids the server-time wait (assumed to be 3 seconds). If the object is in the browser cache, local access avoids the transmission time. If the local access saves both, then the R_local is a fraction of a second. Hence,
R_local = 100ms (browser cache)
= 300ms (proxy cache intranet)
= 2s (proxy cache modem)
= 10s (proxy cache wireless)

Proxy cache studies indicate that *H_proxy_cache = 0.4* is an upper bound [17,18]. Anecdotal evidence suggests browser hit ratios are smaller: assume. *H_browser_cache = 0.20*. Assuming a 20$/hr human cost, each second costs 0.55 cents. Using that number, Table 3 computes the response-time savings using the *Response_Time_Improvement* equation above.

Table 3: Shows the benefits of browser and proxy or client caching (pennies saved) assuming people's time is worth 20$/hr.

connection	cache	R_remote seconds	R_local seconds	H hit rate	People Savings ¢/page
LAN	proxy	3	0.3	.4	0.6
LAN	browser	3	0.1	.2	0.3
Modem	proxy	5	2	.4	0.7
Modem	browser	5	0.1	.2	0.5
Mobile	proxy	13	10	.4	0.7
Mobile	browser	13	0.1	.2	1.4

If a user makes ten requests per hour, and uses the web 400 hours per year then the benefit of caching is between 3 and 14 cents per hour. For our hypothetical user, this is a savings of between $12 and 48 per year. This should be balanced against the cost of the disk to store the pages – but as mentioned earlier, $12 will buy a LOT of disk space. Indeed, our hypothetical user is accessing 4,000 10KB pages that are at most 40 MB. This is less than a dollar's worth of disk space.

Having computed the savings for a cached page (Table 3), we can now compute the point where caching a page begins to pay off. Table 4 has the calculation. The first column of Table 4 estimates download costs from Odlyzko [14 table 7] and assumes a wireless (1KBps) link costs $0.1/minute ($6/hr). The second column of Table 4 assumes desktop disks cost 30$/GB and last 3 years, while mobile storage devices are 30x more expensive.

The break-even cost of storing a page happens when the storage rent matches the download cost. The download cost has two components: the network time (*A* in Table 4) and the people time *C*. The fourth column of the table shows the calculation ignoring people's time, *C*. In that case the break-even interval is a year or more rather than many decades. When people time is included, the reference interval rises to many decades. In either case, the table indicates that caching is very attractive: cache a page if will be referenced within the next 5 years (longer than the lifetime of the system (!)).

Certainly, our assumptions are questionable, but the astonishing thing is that a very wide spectrum of assumptions concludes that a "cache everything" strategy is desirable.

Table 4: Caching is a very good deal: cache web pages if they will be re-used within the few years.

	A $/10 KB download network cost	*B* $/10 KB storage/mo	*Time =A/B* Break-even cache storage time	C People Cost Of download $ (table 3)	*Time=* *(A+C)*/B Break Even
Internet/LAN	1e-4	8.E-06	13 months	0.02	184 years
Modem	2E-4	8.E-06	27 months	0.03	307 years
Wireless	1E-2	2.E-04	44 months	0.07	30 years

How will Table 4 change with time? Network speeds are predicted to increase and network costs are predicted to drop. Column 4, *Time=A/B*, may drop from 10 months to one day. But column 6, *Time=(A+C)/B*, will grow as people's time grows in value, while the cost of technology (*A* and *B*) decline. In summary, technology trends suggest that web page caching will continue be popular, especially for bandwidth-limited mobile devices.

How much would it cost to cache all web accesses for a year? If users make 10 requests per hour with a hit ratio of H=0.4 the cache gets 4 hits and 6 new objects per user hour. For an 8-hour workday and 10KB objects, this adds 480KB per user per day. If H=0.2, then it adds 640KB per user per day. In both cases, this is about a penny a day. So, again we conclude a simple "cache everything" strategy is a good default.

These calculations suggest the simple rule:

22. *Cache web pages if there is any chance they will be re-referenced within their lifetime.*

Web object lifetimes are bi-modal, or even tri-modal in some cases. Studies show median lifetimes to be a few days or few tens of days [19]. The average page has a 75-day lifetime (ignoring the modalities and non-uniform access.) A heuristic that recognized high-velocity pages would both improve usability (by not showing stale cached pages) and would save cache storage. This is a area of active research and development.

A major assumption in these calculations is that server performance will continue to be poor: 3 seconds on average. Popular servers tend to be slow because web site owners are not investing enough in servers and bandwidth. With declining costs, web site owners could invest more and reduce the 3-second response time to less than a second. If that happens, then the web cache's people cost savings will evaporate, and the need for caching would be purely to save network bandwidth and download time -- which we believe will only be a scarce resource for mobile devices.

6. Summary

Data stores will become huge. Our biggest challenge is to make it easy to access and manage them. Automating all the tasks of data organization, accesses, and protection.

Disk technology is overtaking tapes, but at the same time disks are morphing into tape-like devices with primarily sequential access to optimize the use of disk arms. Meanwhile, RAM improvements encourage us to build machines with massive main memory. Indeed, the main change to Amdahl's balanced system law is that alpha (=MIPS/RAM size) is rising from 1 to 10.

Network bandwidth is improving at a rate that challenges many of our design assumptions. LAN/SAN software is being streamlined so it is no longer the bottleneck. This may well allow a re-centralization of computing.

Still, data caching is an important optimization. Disk caching still follows the 5-minute random rule and the one-minute sequential rule. Web caching encourages designs that simply cache all pages.

7. Acknowledgments

We thank Dave Patterson of Berkeley and especially Terrance Kelly of Michigan who both gave us very valuable criticism that improved the presentation and sharpened our arguments.

8. References

[1] IBM UltraStar72, http://www.storage.ibm.com/ hardsoft/diskdrdl/ultra/72zxdata.htm.
[2] Brewster Kahle, private communication, http://archive.org
[3] Data *heat* is the number of times the data is accessed per second.
[4] Dell tpcC: http://www.tpc.org/results/individual_results /Dell/ dell_8450_99112201_es.pdf
[5] HP tpcH: http://www.tpc.org/new_result/h-result1.idc?id=100021501
[6] J. L. Hennessy, D.A. Patterson, *Computer Architecture, a Quantitative Approach*. Morgan Kaufman, San Francisco, 1990, ISBN 1-55860-069-8
[7] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, W. E. Baker, "Performance Characterization Of A Quad Pentium Pro SMP Using OLTP Workloads," ACM ISCA p. 15-26. June 1998.
[8] A. Ailamaki, D. J. DeWitt, M. D. Hill, D. A. Wood. "DBMSs On A Modern Processor: Where Does Time Go?" VLDB 99, pp. 266-277, Sept 1999.
[9] H. Garcia-Molina, A. Park, L.R. Rogers: "Performance Through Memory." ACM SIGMETRICS, Performance Evaluation Review 15(1), May 1987. pp. 122-131.
[10] Hsu, W.H., Smith, A.J., Young, H.C., "I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks -- An Analysis at the Logical Level." TR UCC/CSD-99-1071, UC Berkeley, Nov. 1999.
[11] J. Gray, "The Cost of Messages," ACM PODC, 1988, p1-7
[12] Virtual Interface Architecture: http: //www.viarch.org
[13] G. Gilder, "Fiber Keeps Its Promise: Get ready. Bandwidth will triple each year for the next 25." *Forbes*, 7 April 1997. http://www.forbes.com/asap/97/0407/090.htm
[14] A. M. Odlyzko "The Economics of the Internet: Utility, Utilization, Pricing, and Quality of Service, http://www.research.att.com/~amo/doc/networks.html
[15] R.H. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D.E. Culler, J.M. Hellerstein, D.A. Patterson, "Rivers. Cluster I/O with River: Making the Fast Case Common." IOPADS '99.
[16] J. Gray, G. Graefe, "The 5 minute rule, ten years later," SIGMOD Record 26(4): 63-68, 1997
[17] R. Tewari and M. Dahlin and H M. Vin and J. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet", IEEE ICDCS'99 June, 1999.
[18] A. Wolman and G. Voelker and N. Sharma and N. Cardwell, A. Karlin, H. Levy,"On the scale and performance of cooperative web proxy caching", ACM SOSP'99, pp.16--21, Dec., 1999.
[19] J. Gwertzman, M. Seltzer, "World-Wide Web Cache Consistency," 1996 USENIX Annual Technical Conference, Jan. 1996.
[20] T. Kelley, D. Reeves, "Optimal Web Cache Sizing: Scalable Methods for Exact Solution," Feb. 2000, to appear in 5th Int. Conf on Web Caching and Content Delivery Workshop, 22 May, Lisbon, Portugal.
http://ai.eecs.umich.edu/~tpkelly/papers/wcp.pdf