A Comparison of File System Workloads

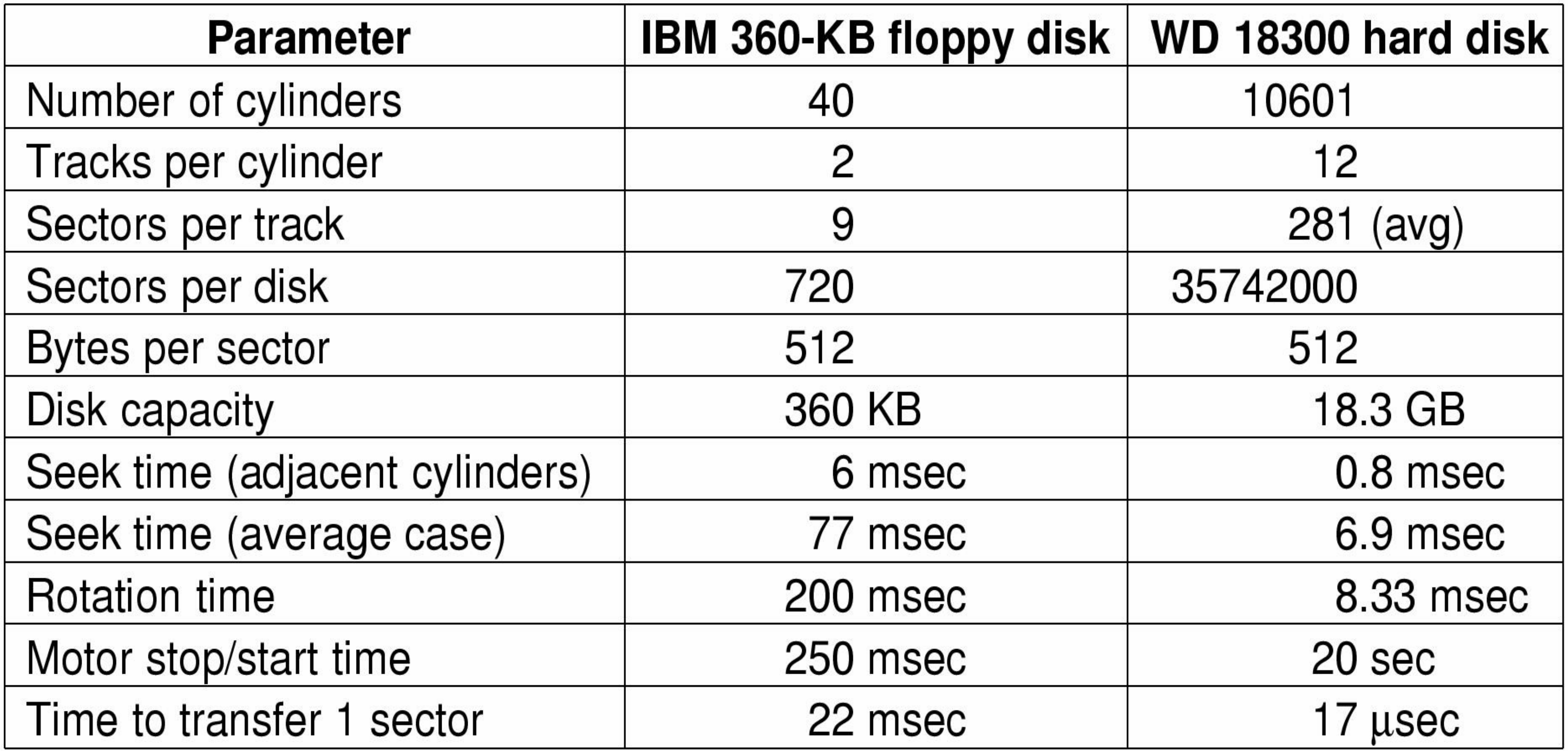
D. Roselli, J. R. Lorch, T. E. Anderson

Proc. 2000 USENIX Annual Technical Conference

File System Performance

- Integral component of overall system performance
- Optimised for common usage patterns
  - Like most components of operating systems

Evolution of Disk Hardware



Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk

Things to Note

- Average seek time is approx 12 times better
- Rotation time is 24 times faster
- Represents a gradual engineering improvement
- Operation becoming more critical as disk latency lags behind processor speed improvement
  - Processor has gone from 4.77 MHz → 1 GHz
  - Approximate 200 times speed up

What Might Be Good To Know To Optimise Performance?

- File size distribution
- File access patterns
  - Sequential, random, in-between
- File lifetime
  - Block lifetime
  - Overwrites
- Proportion of read to writes
  - Does one dominate the other?
- Locality of access to specific files
  - Temporal and Spatial
- How sensitive are the above to workload?
- How do the above characteristics evolve with technology?

How Can We Determine File System Usage Characteristics?

- Static Analysis
  - One approach: examine file system meta-data
  - File Size distribution, File Age
  - Only provides a current snapshot of the system, does not provide access patterns, etc.
- Dynamic Analysis
  - Instrument the kernel to provide file system traces
    - Care must to taken not to perturb the system
  - Can provide much more information about actual system behaviour

Trace Details

- Traced HP-UX and NT 4.0
- Workloads
  - Instructional Environment
    - 8 months (two sessions), 20 machine undergraduate lab
  - Research Environment
    - 1 Year, 13 postgrad and staff machines
  - Web Server
    - 1 month, 2300 hits per day, image server for online library
  - NT
    - Varying length, but 31 days used. 8 Machines, “desktop” work load
- 150 GB compressed traces
- Sprite [Baker et. al.,1991] included for comparison

Basic Results

WEB access significantly more data

Read:Write ratio varies significantly with workload

- stat() represents a significant fraction of all syscalls (INS 42%, RES 71%, WEB 10%, NT 26%)
- Attributed to ‘ls’ and stat() before open()

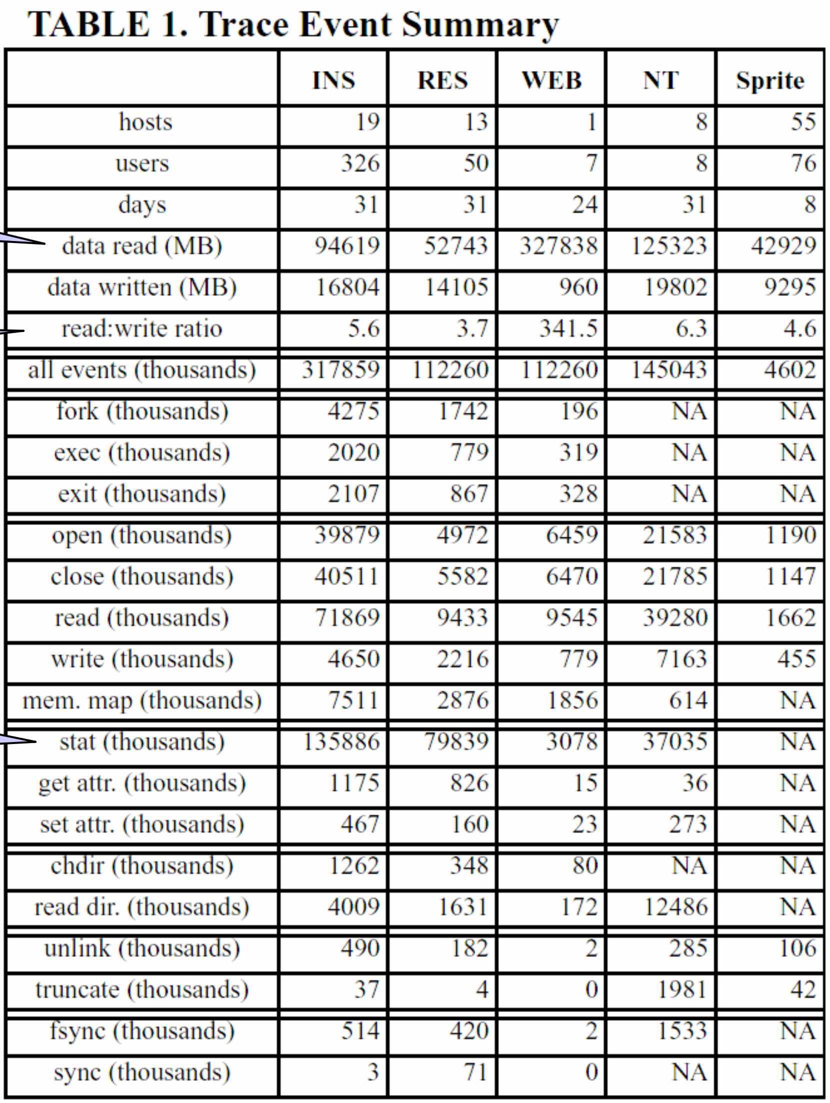


TABLE 1. Trace Event Summary

	INS	RES	WEB	NT	Sprite
hosts	19	13	1	8	55
users	326	50	7	8	76
days	31	31	24	31	8
data read (MB)	94619	52743	327838	125323	42929
data written (MB)	16804	14105	960	19802	9295
read:write ratio	5.6	3.7	341.5	6.3	4.6
all events (thousands)	317859	112260	112260	145043	4602
fork (thousands)	4275	1742	196	NA	NA
exec (thousands)	2020	779	319	NA	NA
exit (thousands)	2107	867	328	NA	NA
open (thousands)	39879	4972	6459	21583	1190
close (thousands)	40511	5582	6470	21785	1147
read (thousands)	71869	9433	9545	39280	1662
write (thousands)	4650	2216	779	7163	455
mem. map (thousands)	7511	2876	1856	614	NA
stat (thousands)	135886	79839	3078	37035	NA
get attr. (thousands)	1175	826	15	36	NA
set attr. (thousands)	467	160	23	273	NA
chdir (thousands)	1262	348	80	NA	NA
read dir. (thousands)	4009	1631	172	12486	NA
unlink (thousands)	490	182	2	285	106
truncate (thousands)	37	4	0	1981	42
fsync (thousands)	514	420	2	1533	NA
sync (thousands)	3	71	0	NA	NA

Block Lifetime

- RES
  - Knee due to netscape database updates
- WEB
  - Knee due to updates logs and database back-end
- sprite - unknown
- INS and NT “flat”

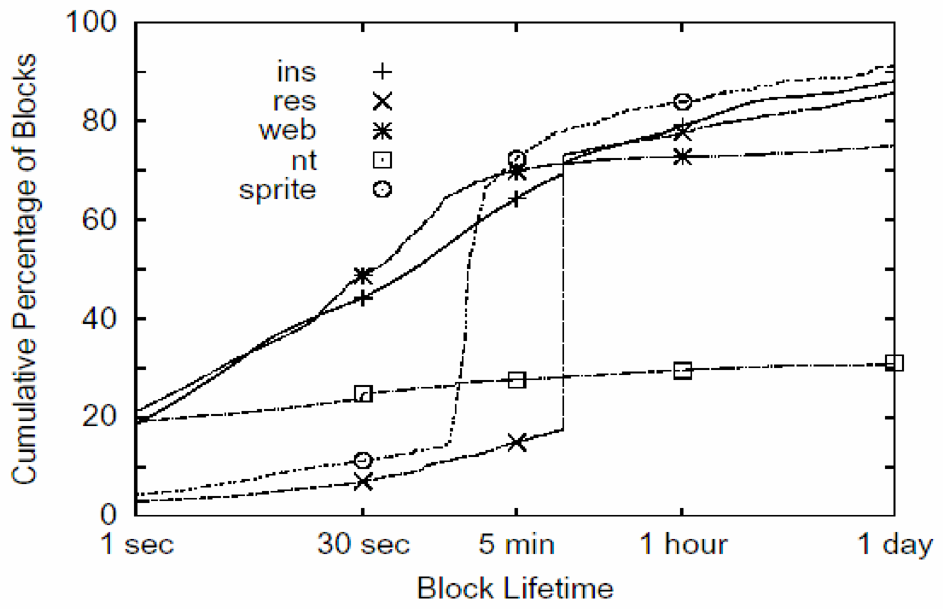


FIGURE 2. Block Lifetime. This graph shows create-based block lifetimes using a block size of 512 bytes. Points demarcate the 30 second, 5 minute, and 1 hour points in each curve. The end margin is set to 1 day for these results.

Block Lifetime

- Most blocks die due to overwrites
  - INS 51%, RES 91%, WEB 97%, NT 86%
- Small number of files overwritten repeatedly
  - INS 3%, 15 times each
  - RES 2%, 160 times
  - WEB 5%, 6300 times
  - NT 2%, 251 times
- Observations
  - File block life-time > 30 second 'sync' time
  - Locality of overwrite may provide opportunity

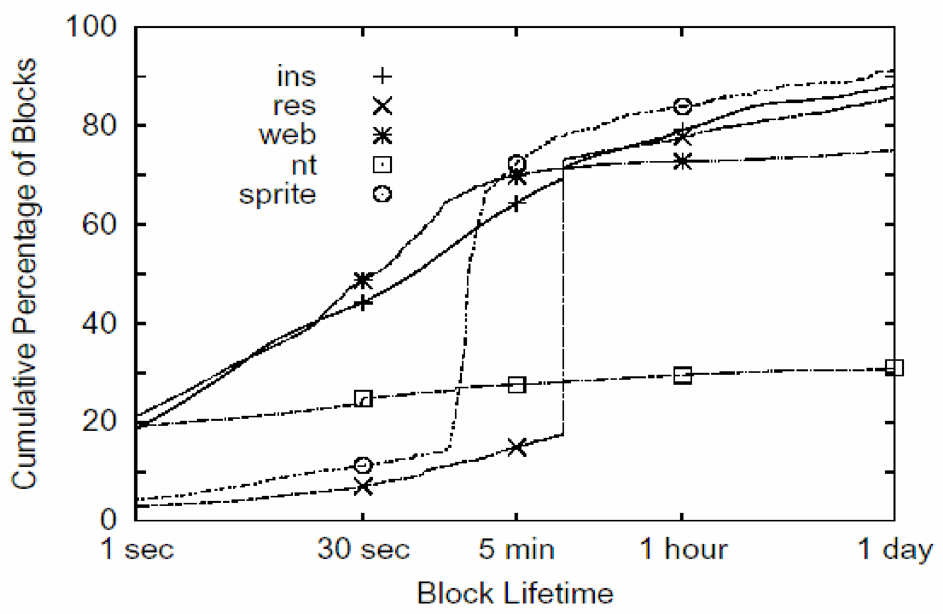


FIGURE 2. Block Lifetime. This graph shows create-based block lifetimes using a block size of 512 bytes. Points demarcate the 30 second, 5 minute, and 1 hour points in each curve. The end margin is set to 1 day for these results.

Bandwidth Versus Write Delay

- Write delay must be significantly greater than 30 sec
  - Need NVRAM
- Has little affect on some workloads
  - affect directly related to file life times
- Simulated 4MB, 16MB, and infinite buffer with similar results
  - Small (4MB) NVRAM sufficient for 1 day for workloads examined

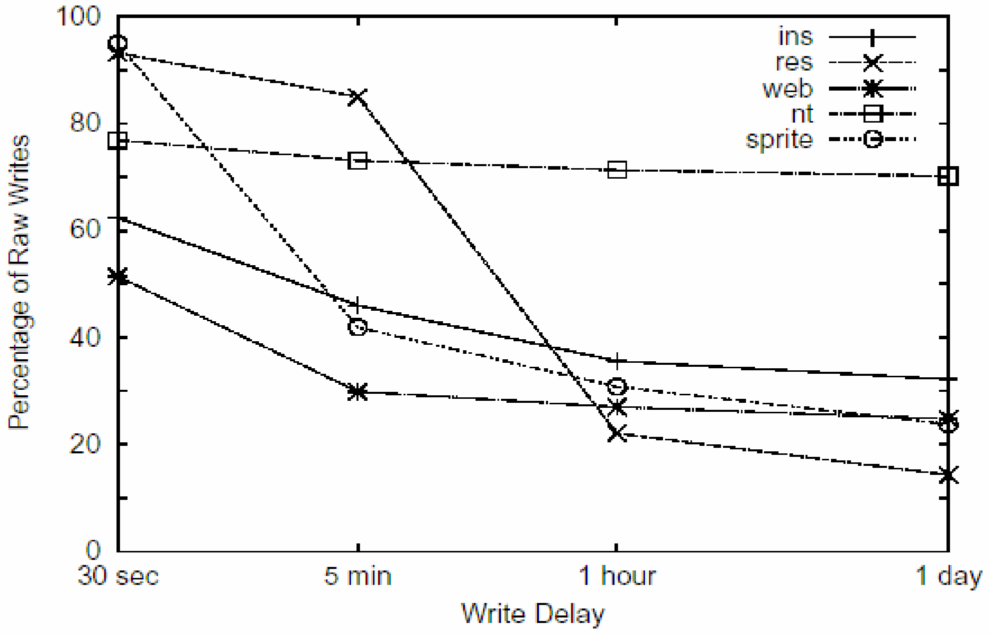


FIGURE 3. Write Bandwidth versus Write Delay. Using a simulated 16MB write buffer and varied write delay, we show the percentage of all writes that would be written to disk. For these results, we ignore calls to `sync` and `fsync`.

Read Reduction Via Caching

- Small caches absorb significant number of block read requests
- Diminishing returns as cache size increases
- WEB has large working set size

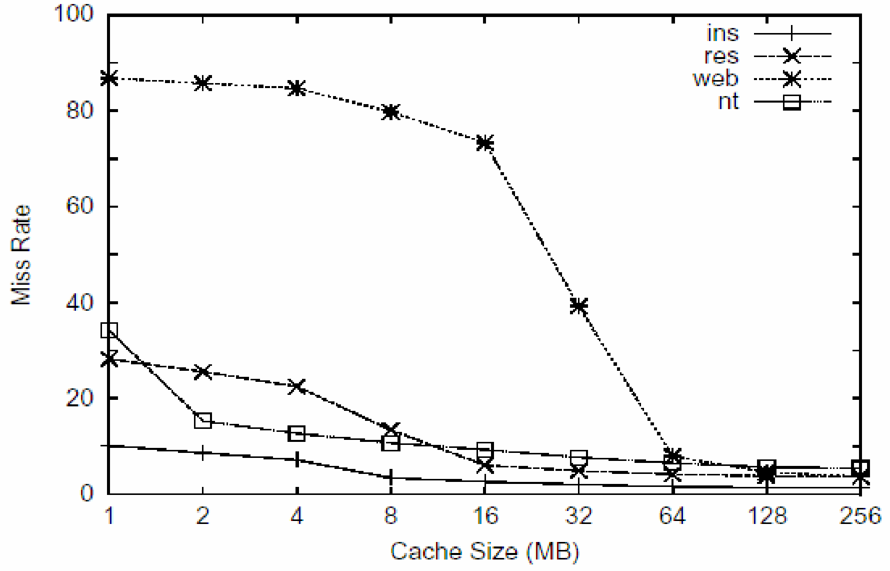


FIGURE 4. Read Bandwidth versus Cache Size. This graph shows the percentage of all block read requests that miss the cache versus cache size. The block size used by the cache simulator is 4KB. The cache was warmed with a day of traces before generating results.

Alternative Metric

- Motivation
  - Disk bandwidth is improving much faster than seeks times
  - Seek-causing cache misses more critical than simple block misses
- File Read misses
  - If a block cache miss is to the same file as previous miss it’s ignored.
    - Assumed to be consecutive block ⇒ no seek
  - If it’s to a new file, then we have a *file read miss*
- File Write misses can be defined similarly

File Reads versus Cache Size

- Small cache size reduce file reads significantly
- Note: WEB has fewer file misses than block misses
  - Block misses within larger file

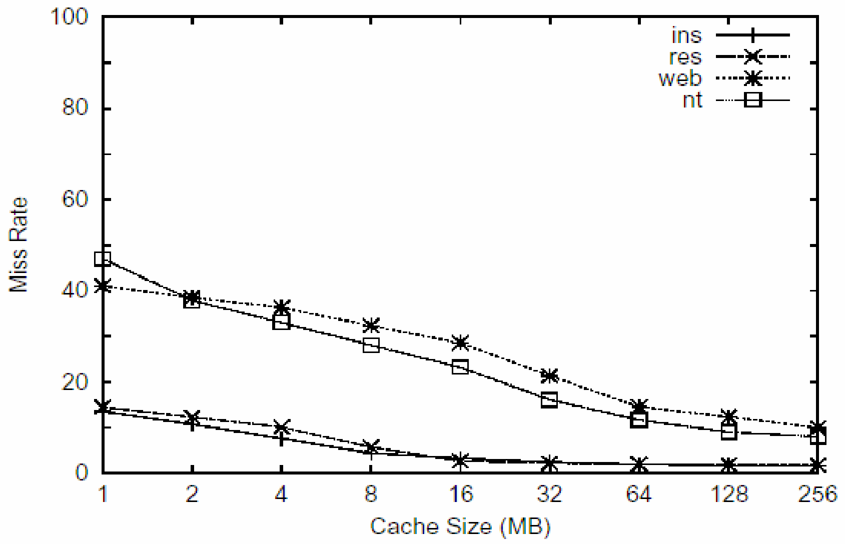


FIGURE 5. File Reads versus Cache Size. The miss rate is the percentage of file read misses out of the raw number of file reads. This graph shows the file miss rate for various cache sizes. The block size used by the cache simulator is 4KB. The cache was warmed with a day of traces before results were collected.

Reads or Writes Dominate?

- Dominating reads suggests BSD like file system structure
- Dominating write suggests log-structured file system
- Writes depend on write-delay; Reads depend on cache size
- Workload and environment play significant role ⇒ in general we must consider both reads and writes

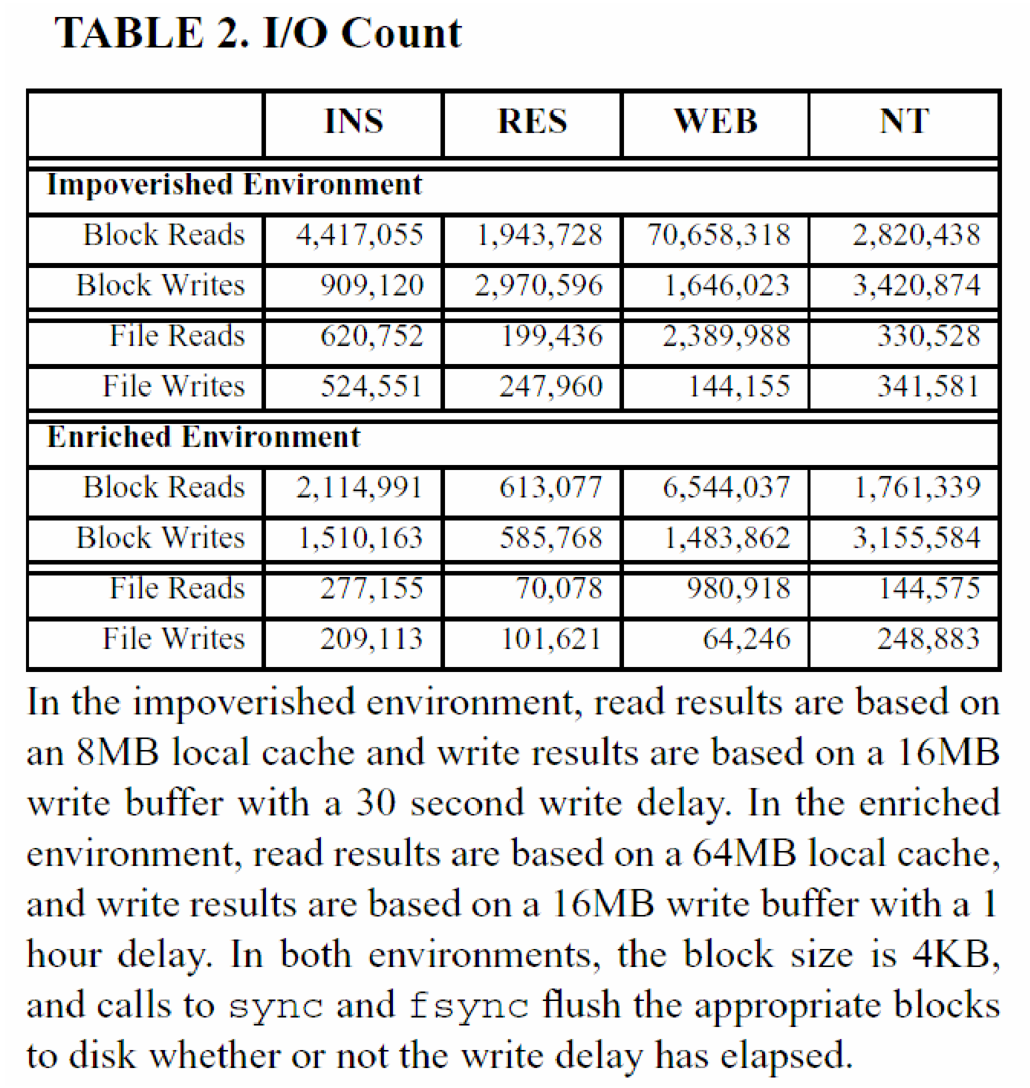


TABLE 2. I/O Count

	INS	RES	WEB	NT
Impoverished Environment				
Block Reads	4,417,055	1,943,728	70,658,318	2,820,438
Block Writes	909,120	2,970,596	1,646,023	3,420,874
File Reads	620,752	199,436	2,389,988	330,528
File Writes	524,551	247,960	144,155	341,581
Enriched Environment				
Block Reads	2,114,991	613,077	6,544,037	1,761,339
Block Writes	1,510,163	585,768	1,483,862	3,155,584
File Reads	277,155	70,078	980,918	144,575
File Writes	209,113	101,621	64,246	248,883

In the impoverished environment, read results are based on an 8MB local cache and write results are based on a 16MB write buffer with a 30 second write delay. In the enriched environment, read results are based on a 64MB local cache, and write results are based on a 16MB write buffer with a 1 hour delay. In both environments, the block size is 4KB, and calls to `sync` and `fsync` flush the appropriate blocks to disk whether or not the write delay has elapsed.

How Common Are Memory Mapped Files?

- Shared libraries are common
- Counted number of processes that read(), write(), mmap().
- Memory mapped files are commonplace and need to be considered in system design.

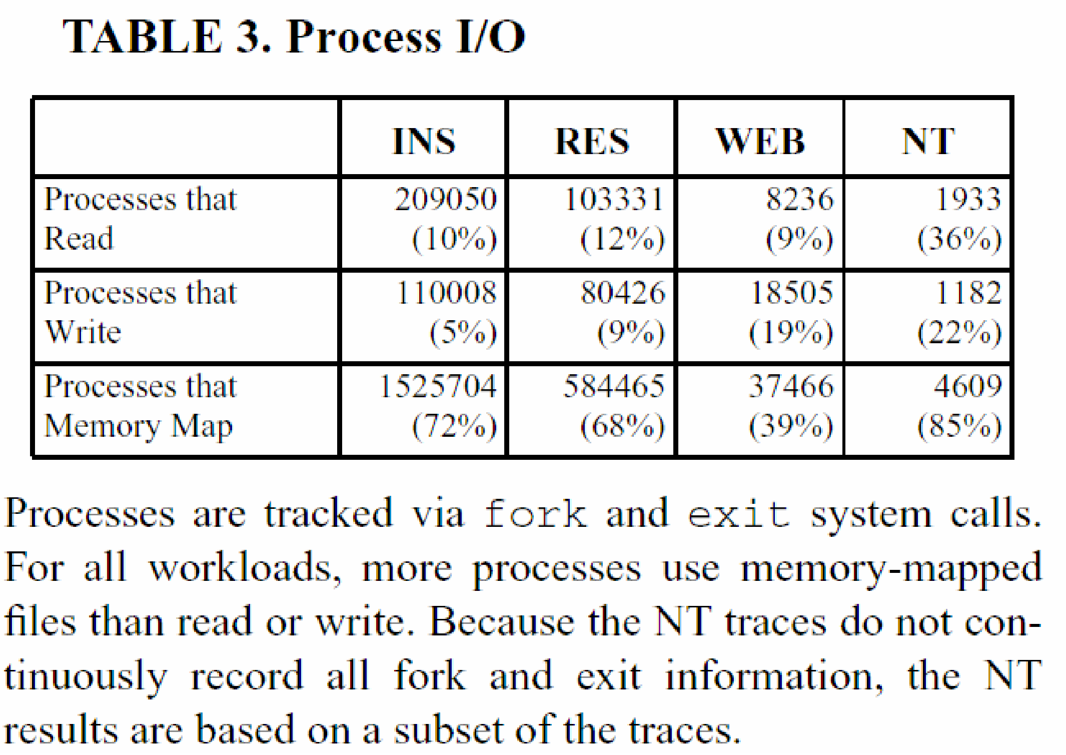


TABLE 3. Process I/O

	INS	RES	WEB	NT
Processes that Read	209050 (10%)	103331 (12%)	8236 (9%)	1933 (36%)
Processes that Write	110008 (5%)	80426 (9%)	18505 (19%)	1182 (22%)
Processes that Memory Map	1525704 (72%)	584465 (68%)	37466 (39%)	4609 (85%)

Processes are tracked via `fork` and `exit` system calls. For all workloads, more processes use memory-mapped files than read or write. Because the NT traces do not continuously record all fork and exit information, the NT results are based on a subset of the traces.

Memory Mapped File Usage

- Large number of mmap() calls, but small number of files mapped
  - Small number of file access repeatedly
- High degree of concurrent access ensures low cache miss rate
  - File evicted if not currently in use
- Shared libraries

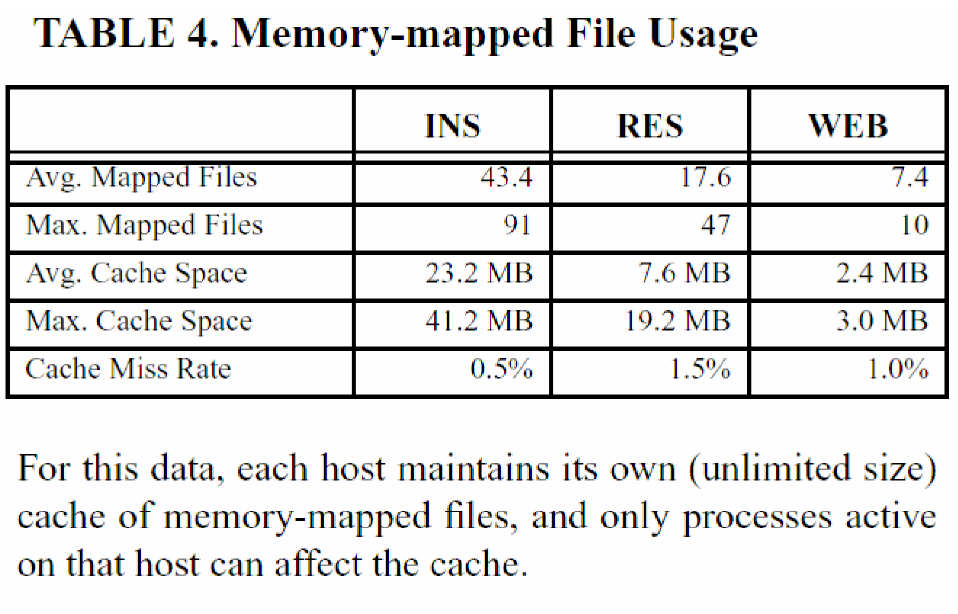


TABLE 4. Memory-mapped File Usage

	INS	RES	WEB
Avg. Mapped Files	43.4	17.6	7.4
Max. Mapped Files	91	47	10
Avg. Cache Space	23.2 MB	7.6 MB	2.4 MB
Max. Cache Space	41.2 MB	19.2 MB	3.0 MB
Cache Miss Rate	0.5%	1.5%	1.0%

For this data, each host maintains its own (unlimited size) cache of memory-mapped files, and only processes active on that host can affect the cache.

Dynamic File Size

- Size recorded on every close
  - Repeated accesses contribute to distribution
- Small files still contribute to workload
  - 88% INS, 60% RES, 63% WEE 24% NT < 16KB
- Compared to sprite, modern workloads have increased in size
  - Max sizes: Sprite 38M, 244MB WEB, 419MB (INS and NT)

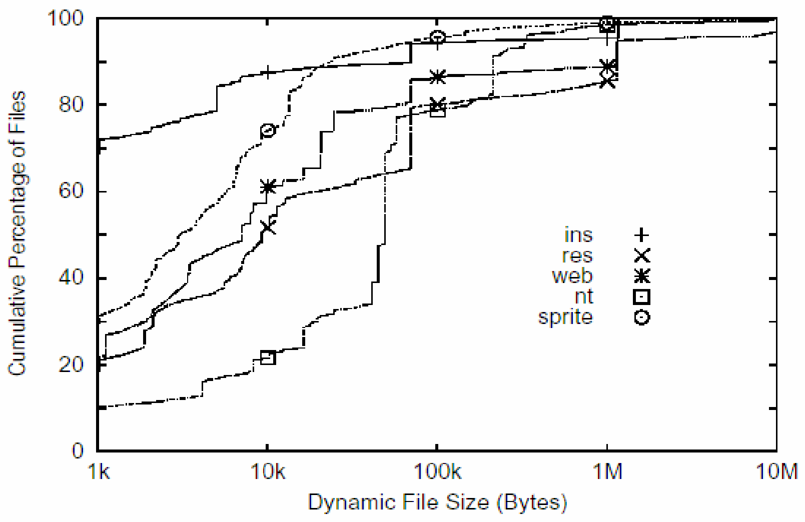


FIGURE 6. Dynamic File Size. We record file size for each accessed file when it is closed. If a file is opened and closed multiple times, we include the file in the graph data multiple times. Points depict sizes of 10KB, 100KB, and 1MB.

Unique File Size

- Each file counted only once in trace
  - Represents range of file size on disk actively accessed
- Note WEB access many small files
  - Small bitmaps
- While file size distribution has not increase significantly, more access to large files (see previous graph)
  - More accesses via indirect blocks
  - Change inode structure?

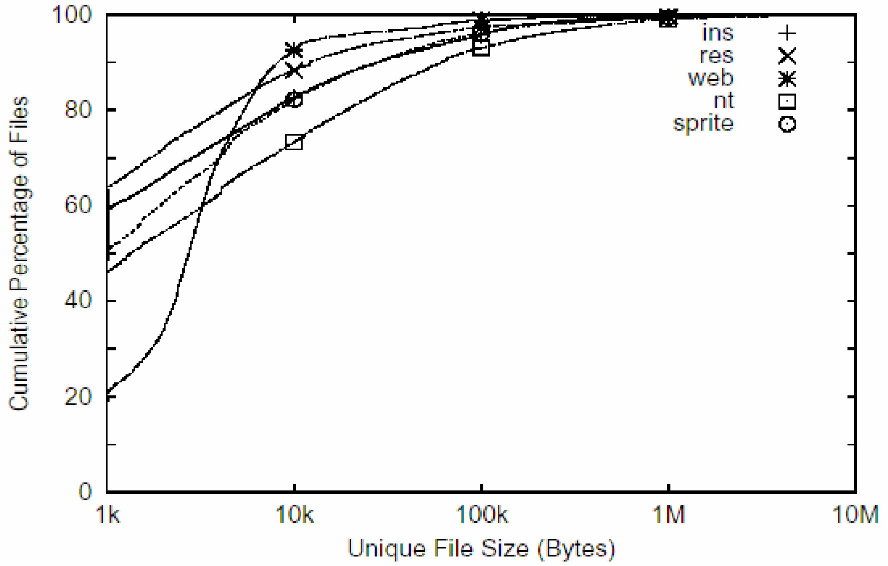


FIGURE 7. Unique File Size. We record file size at the time of file close. If a file is opened and closed multiple times, we only use the last such event in the graph data. Points depict sizes of 10KB, 100KB, and 1MB.

File Access Patterns

- Definitions
  - A *run* is all access to a file between open() and close()
  - A run can be:
    - *Entire*: read/written from start to finish
    - *Sequential*: read/written sequentially, but not beginning to end
    - *Random*: the rest

File Access Patterns

Most runs are read-only with the majority *entire*

Small number of write-only runs with the majority *entire*

Very small number of read-write runs

Most read-write runs are random

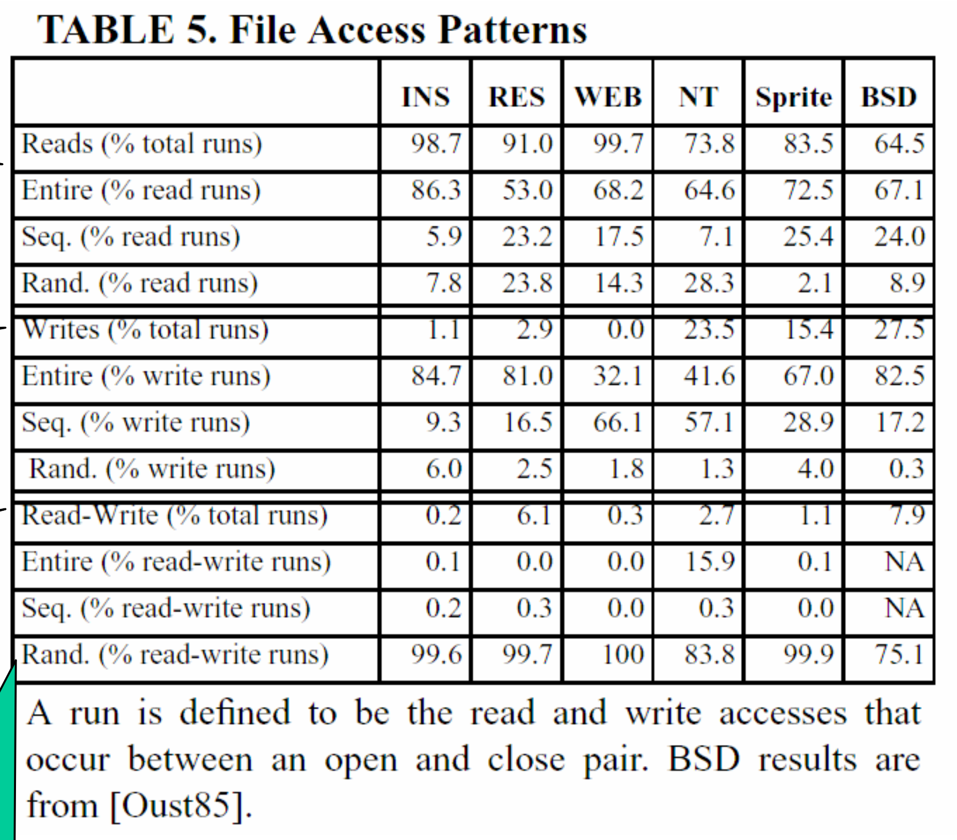


TABLE 5. File Access Patterns

	INS	RES	WEB	NT	Sprite	BSD
Reads (% total runs)	98.7	91.0	99.7	73.8	83.5	64.5
Entire (% read runs)	86.3	53.0	68.2	64.6	72.5	67.1
Seq. (% read runs)	5.9	23.2	17.5	7.1	25.4	24.0
Rand. (% read runs)	7.8	23.8	14.3	28.3	2.1	8.9
Writes (% total runs)	1.1	2.9	0.0	23.5	15.4	27.5
Entire (% write runs)	84.7	81.0	32.1	41.6	67.0	82.5
Seq. (% write runs)	9.3	16.5	66.1	57.1	28.9	17.2
Rand. (% write runs)	6.0	2.5	1.8	1.3	4.0	0.3
Read-Write (% total runs)	0.2	6.1	0.3	2.7	1.1	7.9
Entire (% read-write runs)	0.1	0.0	0.0	15.9	0.1	NA
Seq. (% read-write runs)	0.2	0.3	0.0	0.3	0.0	NA
Rand. (% read-write runs)	99.6	99.7	100	83.8	99.9	75.1

A run is defined to be the read and write accesses that occur between an open and close pair. BSD results are from [Oust85].

File Access Patterns

- Small files (< 20Kb) usually read entirely
- Larger files usually accessed randomly
- Implication for prefetching strategies

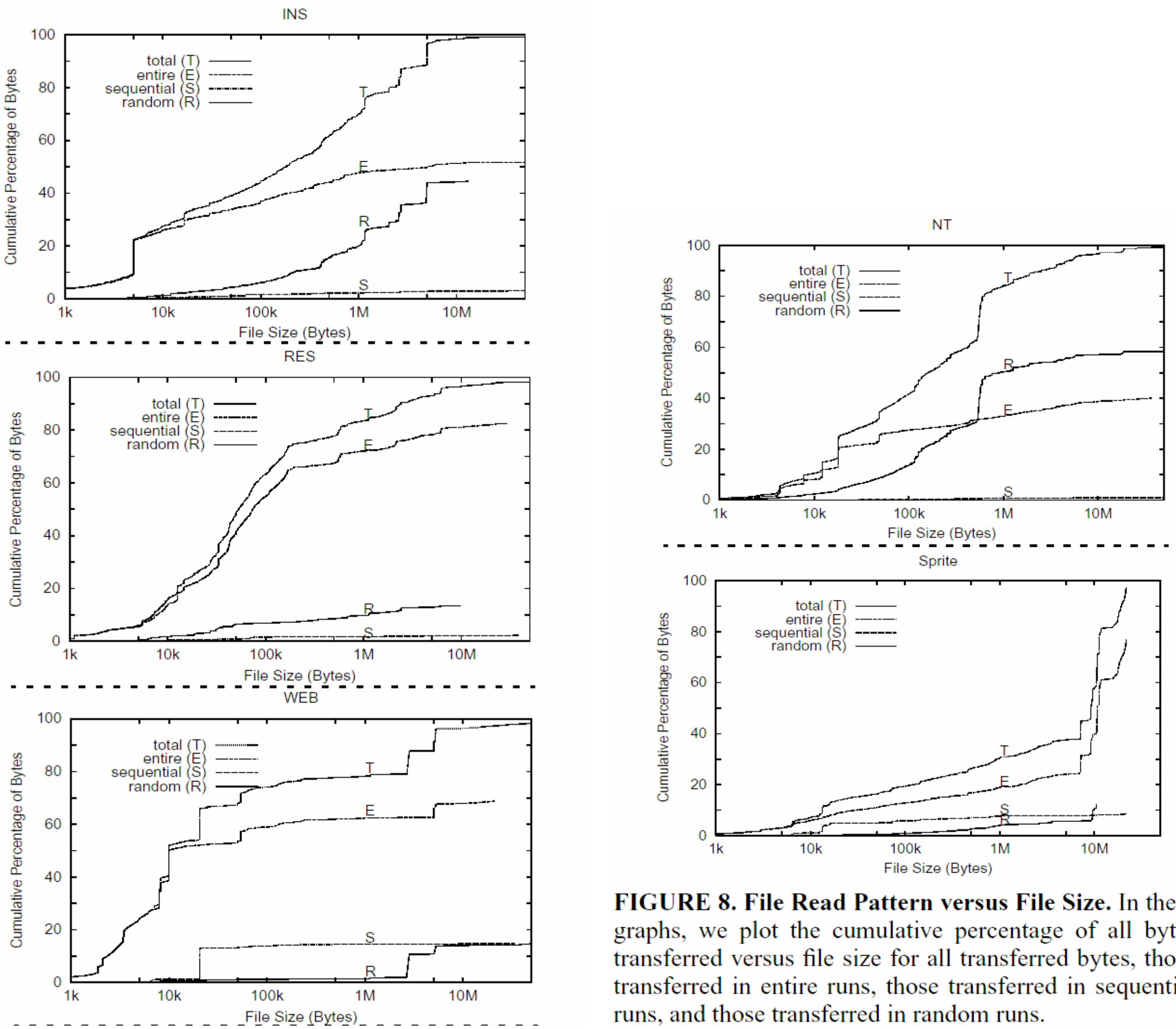


FIGURE 8. File Read Pattern versus File Size. In these graphs, we plot the cumulative percentage of all bytes transferred versus file size for all transferred bytes, those transferred in entire runs, those transferred in sequential runs, and those transferred in random runs.

Read Write Patterns

- For repeatedly accessed file, we calculate percentage of read-only runs
- Bi-modal distribution
  - Tendency to read-mostly or write-only
  - More frequently accessed files more strongly read-mostly or write-mostly

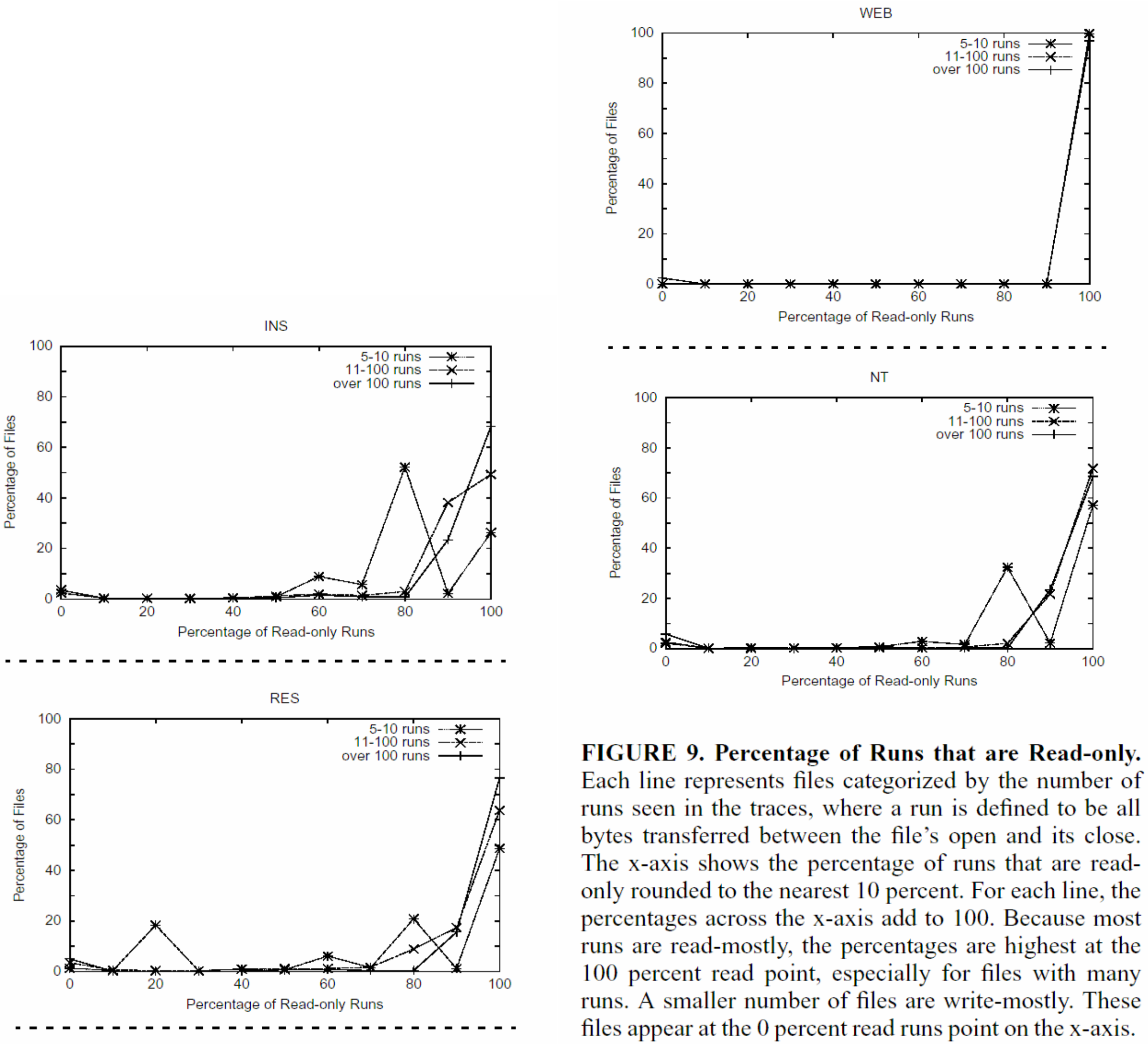


FIGURE 9. Percentage of Runs that are Read-only. Each line represents files categorized by the number of runs seen in the traces, where a run is defined to be all bytes transferred between the file's open and its close. The x-axis shows the percentage of runs that are read-only rounded to the nearest 10 percent. For each line, the percentages across the x-axis add to 100. Because most runs are read-mostly, the percentages are highest at the 100 percent read point, especially for files with many runs. A smaller number of files are write-mostly. These files appear at the 0 percent read runs point on the x-axis.

Conclusions

- Different systems show significantly different I/O loads
- Average block lifetime varied significantly across workloads
    - UNIX: most blocks die within an hour
    - NT: blocks surviving longer than a second survive greatar than a day
  - Most block die from overwrites, overwrites show substantial locality
    - Small write-buffer is sufficient
  - Ideal write-delay varies between workloads and can be greater than 30 seconds

Conclusions

- Caching reduces read traffic
  - Small caches can decrease traffic
  - Diminishing return for increasing cache size
  - Results don’t support “writes dominate for large caches”
- Memory mapping is common
  - Using “not currently in use” as eviction policy results in low cache miss rate
- Larger files accessed compared to previous studies
  - Larger files tend to be accessed randomly
    - Pre-fetching unhelpful