## Measurements of a Distributed File System

(Baker, Hartman, Kupfer, Shirriff, Ousterhout), SOSP'92

- Sprite DFS
- Importance of measurements
- Guide designs
- Two issues:
  - comparison with BSD (1985) study of user-level access patterns (throughput and file sizes)
  - behavior of Sprite's main-memory file caches



- BSD study
  - sequential accesses
  - lifetimes less than 2 seconds
  - short file, but majority of data from a few large ones
  - access rates: few hundred bytes per second
  - guided network file systems design
- This study: Sprite
  - 40 workstations, 4 file servers, diskless clients
  - single file system image, no local disks
  - throughput almost 20 times higher than BSD!
  - larger files
  - variations in cache sizes (large)
  - cache consistency implementation ("perfect")
  - process migration



- Tracing: log info about file system events: opens, closes, deletions
- `lseek` calls
- Tracing at the file servers
- Eight 24-hours periods
- Kernel level: 50 counters
- Table 1



- User activity (table 2)
- File access patterns (table 3 and figures 1, 2, 3)
- File lifetimes (figure 4)



## File Cache Measurements

- Cached on both client and server
- Cache sizes vary depending on needs of virtual memory system
- Writes are delayed by 30 seconds (every 5 seconds daemon checks for dirty data and first writes them to disk, then to cache)
- Cache consistency through:
  - timestamps (when file is opened: server flushes state data from its cache)
  - recall (server recalls dirty data and updates the new client if the writer has not updated server yet)
  - cache disabling (if file open for concurrent write-sharing)



- Size of Sprite caches?
- How much of read and write traffic is absorbed by caches?
- Data lifetime in cache?
- How often does write-sharing happen?
- Best cache consistency algorithm?



## Comparison of Cache Consistency

- *concurrent write-sharing*: if one of the files is opened for writing, then Sprite disables client caching
- slight modification: if concurrent write-sharing is eliminated, then file becomes cacheable
- *token-based* mechanism used by Locus, Echo, etc: a file is always cacheable on at least one client (read-only and read-write token)
- simulation results