A light-weight, cooperative temporary file system (CTFS) for cluster-based Web servers

Jun Wang

Computer Science and Engineering Dept.

University of Nebraska Lincoln

Outline

- Introduction
- CTFS design
- Experimental methodology
- Simulation results
- Related work
- Conclusion

Introduction

- A commodity cluster-based Web server becomes more popular
- Disk I/O is an important performance bottleneck in non-content aware cluster-based Web servers
- Content-aware cluster-based Web servers can alleviate I/O overhead but with complicated implementation; more importantly, they are not helpful for data-intensive Web loads
- Pai et al. (ASPLOS'98) concluded that cluster-based Web servers have disk-bound problems rather than CPU-bound
- CDN and reverse proxy server can not eliminate I/O problem

Current solutions

- The general-purpose native file system (FFS, UFS, etc.) where cluster-based Web servers are designed to run has following problems
  – Local file systems on host server nodes do not share and exchange files, and result in repeated I/Os
  – Specialized distributed file systems have poor portability
  – Multiple buffering or data copies for intra-cluster communications
  – Small docs dominate Web server access pattern but FFS is not good
  – Read requests dominate Web server access pattern
  – Associated access locality is not exploited (Ex., a HTML file with hyper linked files)
  – Meta-data overhead (Ex., frequent access time update)

Our new solution

- A light-weight, cooperative temporary file system (CTFS)
  - Deploy a TFS (temporal file system published in Wang et al's MASCOTS'03) on each server node
    - Manage own data and meta-data on raw disk, bypass native file system (NaFS, for example, FFS, UFS) and adopt customized optimization cache replacement algorithm for Web server workload
    - Work in front of NaFS
    - Store secondary copies and optimize data layout
    - Transparent to users
  - Each TFS peer works cooperatively together with other TFS peers and construct a peer-to-peer TFS software architecture

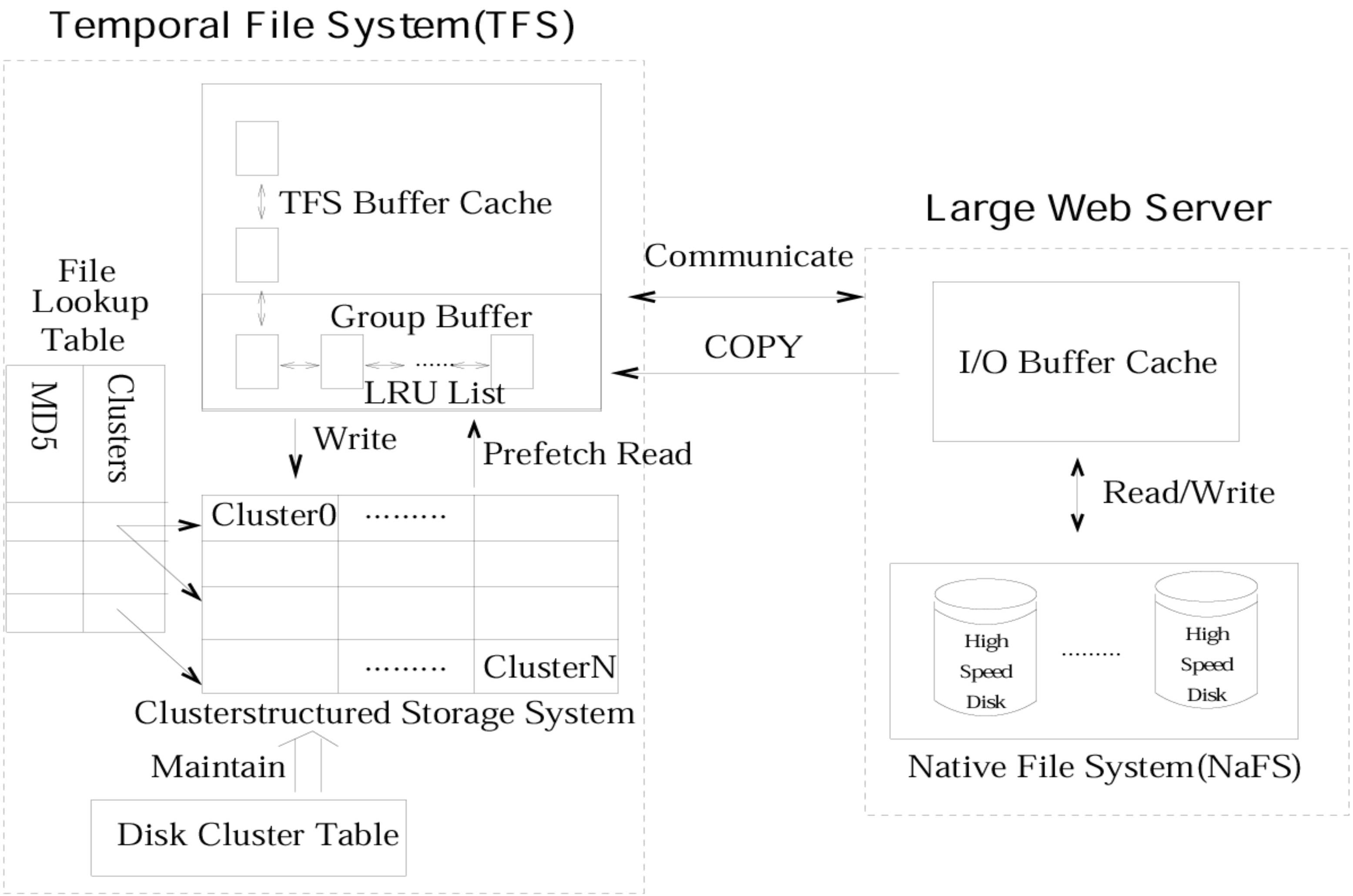
Our new solution (cont.)

- Intra-cluster communication technique adopts remote direct memory access (RDMA)
  - Advantages: low CPU overhead, zero copy, fast transfer and follow user-level implementation
    - Transferring 32 KB data by RDMA takes 0.29 ms while local disk access needs 8 ms. Compared to TCP, RDMA improves performance as much as 29% (Carrera et al. HPCA8)
  - Several successful implementations such as Myrinet-VI, Giganet VIA, M-VIA, ServerNet VIA, Firm VIA
  - Direct Access File System (DAFS) employs VIA/RDMA as network communication technique

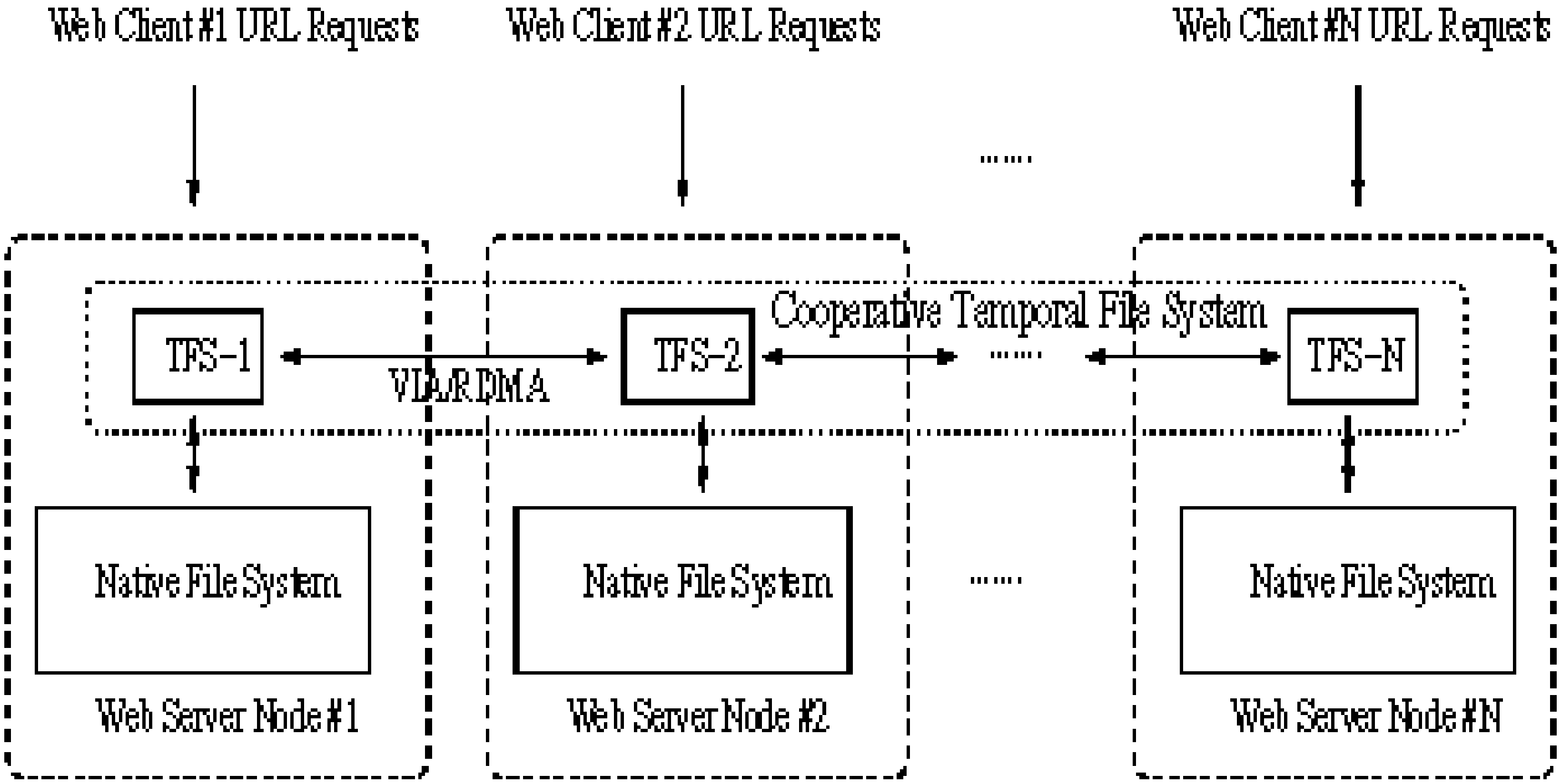
Our new solution (cont.)

- Disadvantages
  - Additional space for second copy, but disk space is too cheap and such a trend will continue. Moreover, CTFS controls the size of its partition by managing only frequently-accessed files

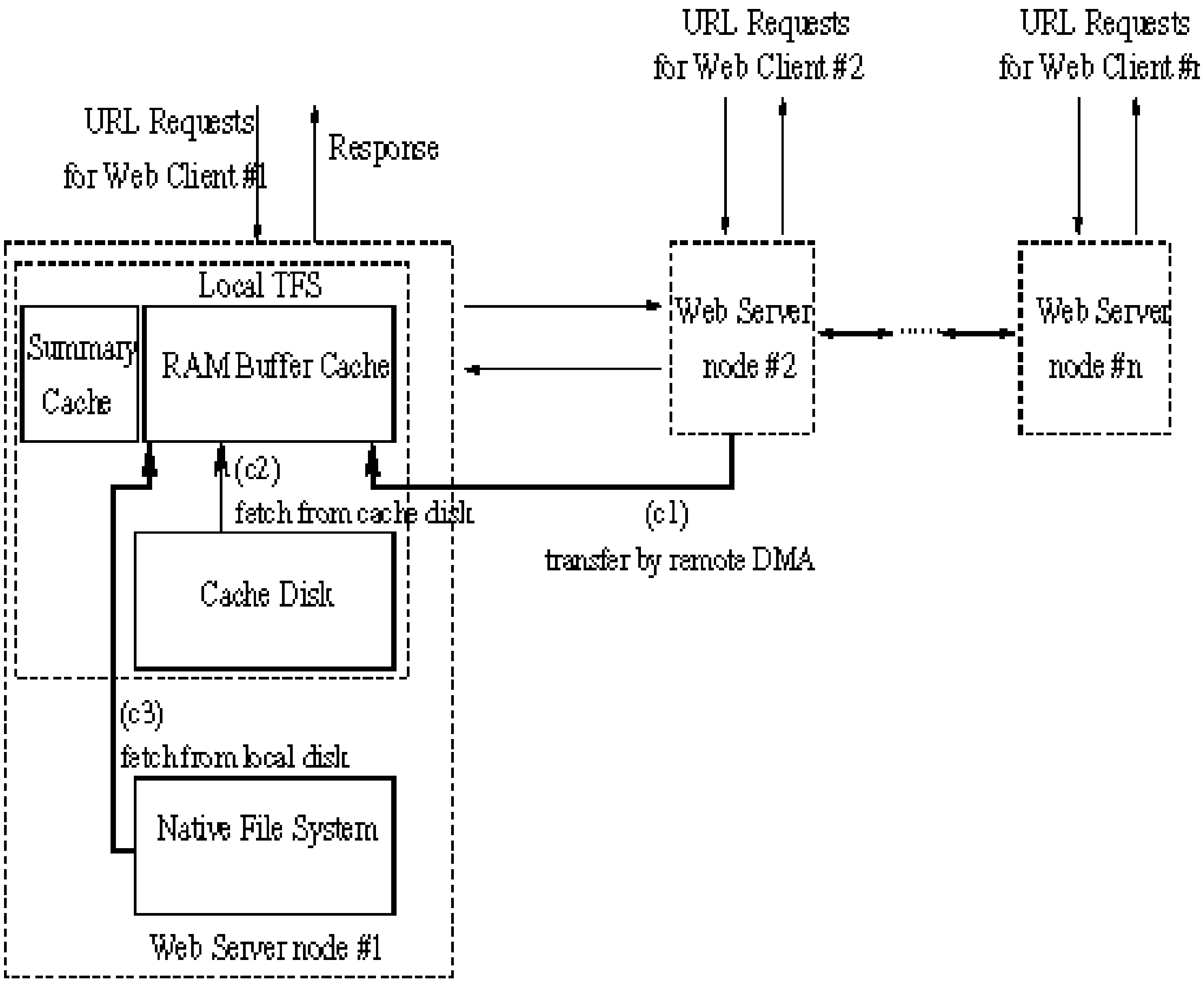
TFS system architecture on a single server node



CTFS for cluster-based Web server architecture



CTFS Cooperative Caching Architecture



Summary Cache Design

- Each TFS peer adopts a bloom filter representing all CTFS files as an in-memory file lookup table
- Similar to summary cache project (Li, Cao and Almeida’s SIGCOMM'98)
- CTFS can support 64 peer TFS with each containing 10 K files, and 1.4 MB memory space for 640 K files in total

Advantageous Prefetch

- Do prefetch simultaneously when reading the target file from remote buffer cache or local CTFS cache disk
- Capture associative locality in Web access load for both local and remote nodes

Consistency Control

- Multiple copies of the same file may be cached on different TFS
- Native file system informs CTFS a stale file, every related TFS invalidate the entire disk cluster that contains the stale file
- Associate access locality indicates the files reside on a cluster may be invalidated together

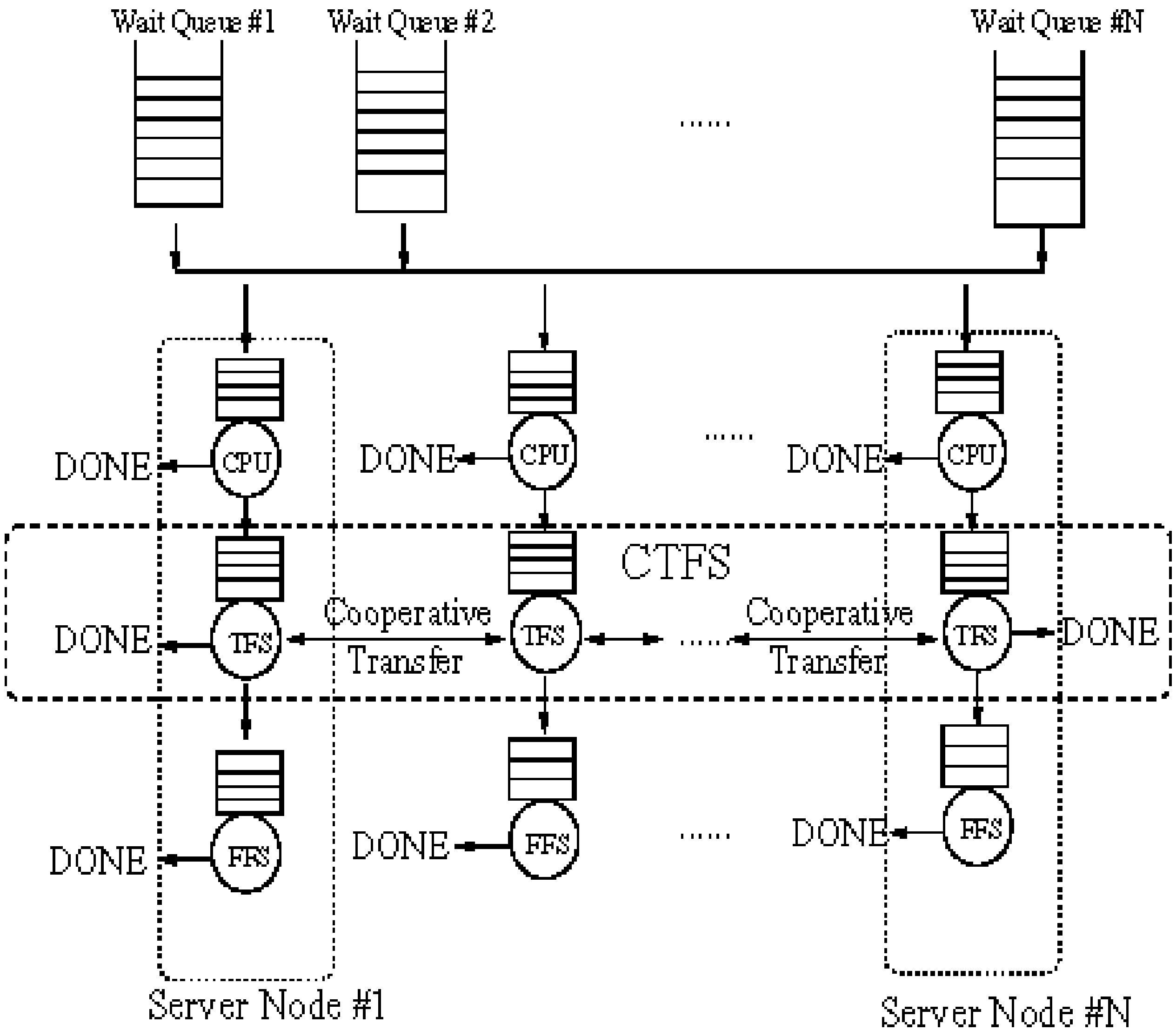
Crash Recovery

- Not a major issue since CTFS is a temporary, read-only file system
- Can implement a journaling file that will log the most-recently-accessed cluster Ids
- After a crash, the clusters on such a log will be used to initialize CTFS buffer cache. Therefore CTFS can achieve a hit rate close to that before the crash

Experimental methodology

- CTFS is a HTTP trace-driven simulator ported from distributed Web server simulator by University of Saskatchewan
- CTFS contains
  - A cluster-based Web server component
  - CTFS together with FFS-async component
  - Disk simulation slaver
- Baseline system: UNIX FFS mounted on asynchronous mode (FFS –async)

CTFS Simulation Model



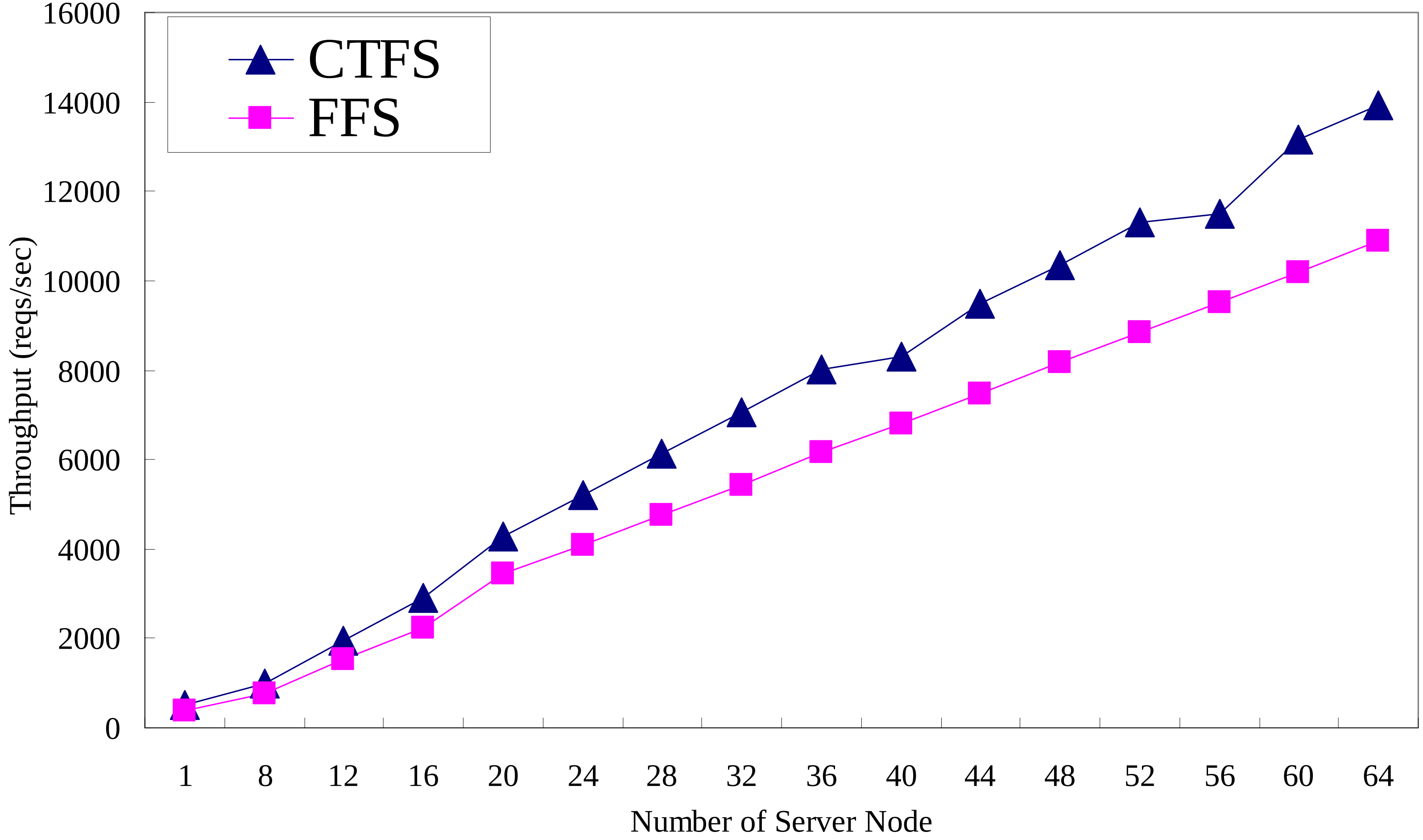
Configure Simulation

- RDMA-based intra cluster communication:
  - 4 byte message takes 9 microseconds, 32 KB message takes 0.29 ms
- Connection establishment and shut down costs are 40 us
- I/O buffer cache replacement policy are Greedy-dual-size
- A weighted round-robin request distribution strategy
- I/O buffer cache size is 512 MB

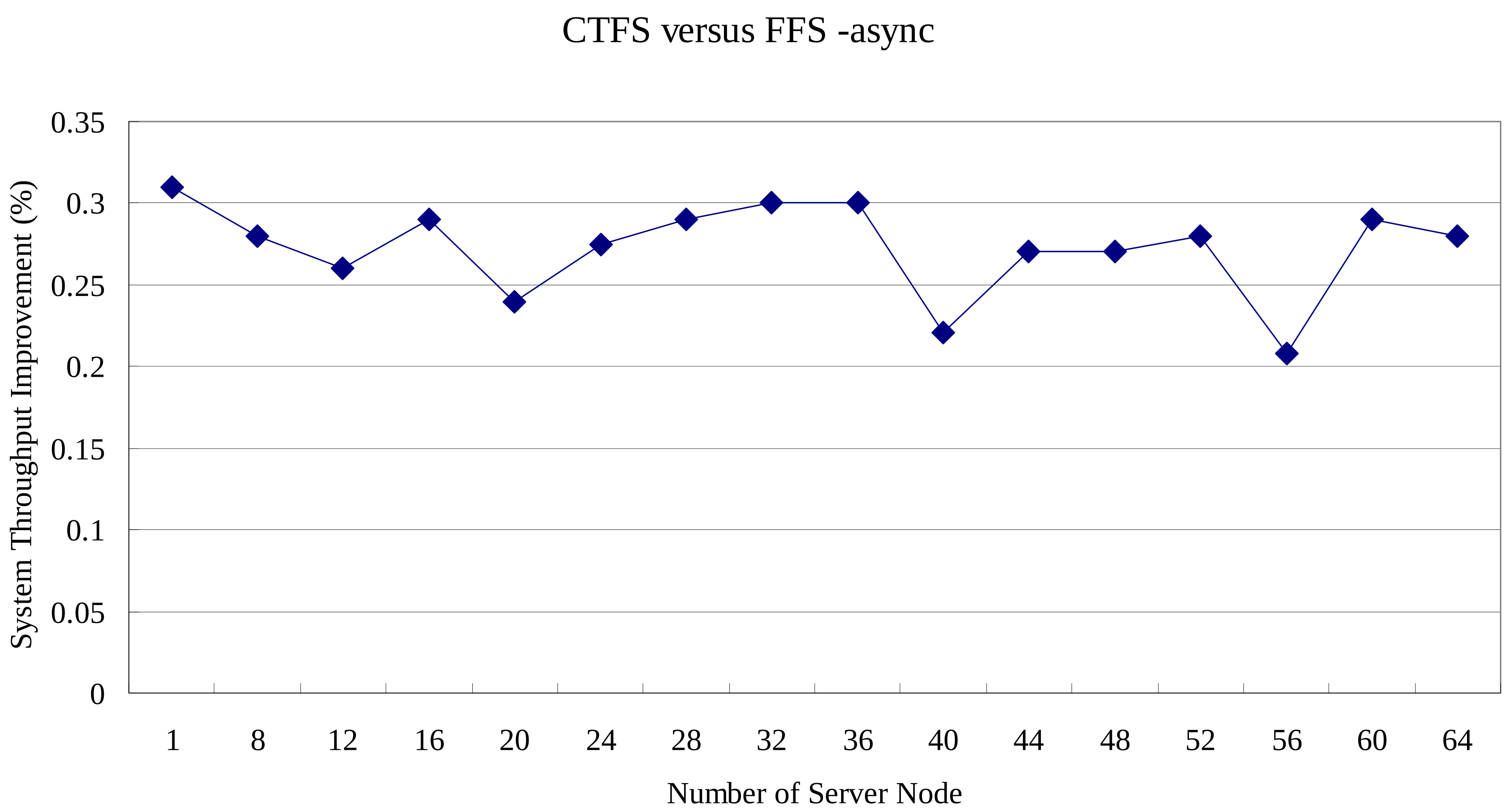
Real-world Workloads

Properties	UCBSep01	SpecWeb99
Location	U. of California Berkeley	U. of Nebraska Lincoln
Duration	Sep.1-Dec.1, 2001	Sep.7-Sep.8 2002
Requests	14,413,271	40,000,000
Bytes Xerred(MB)	320,056	350,000
Unique Bytes(MB)	14,605	25,000
Mean File(B)	26,040	18,584
Median File(B)	6,912	5,094

Experimental Results with Berkeley Trace

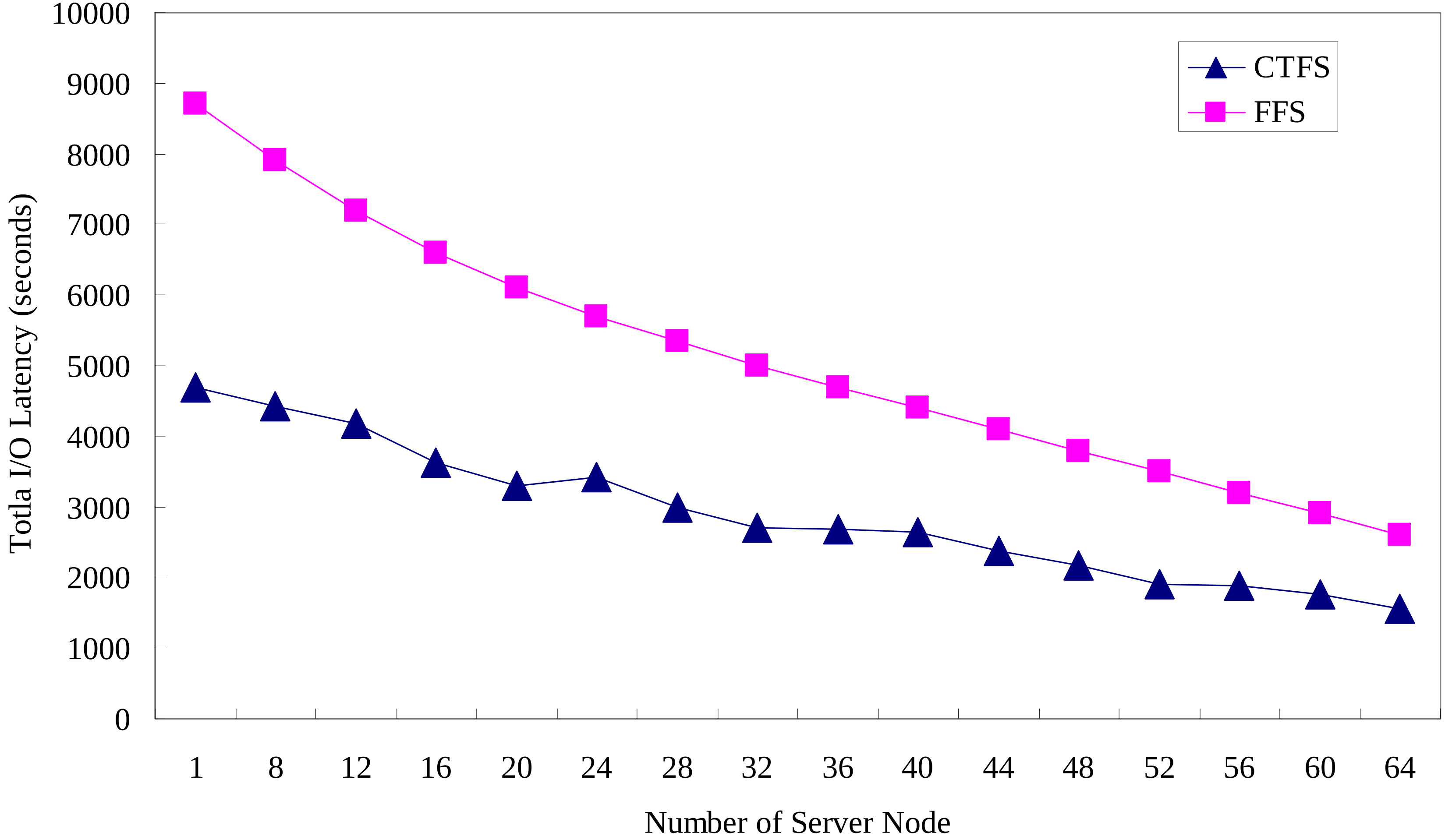


Experimental Results with Berkeley Trace



Computer Architecture and Storage System Research Laboratory (CASS)

Experimental Results with Berkeley Trace



Computer Architecture and Storage System Research Laboratory (CASS)

Related work

- UCFS (Wang & Hu, IEEE TC 2002)
  - UCFS for Web proxy server, both read and write intensive
- TFS for Web server, read intensive (Wang & Li, IEEE MASCOTS 2003)
- Hummingbird (Shriver USENIX'01)
  - TFS no garbage collection overhead
- Other temporary file systems work in kernel space, rather than user space

Conclusions

- CTFS is a new light-weight, cooperative temporary file system for cluster-based Web servers
- Advantages are good portability, scalability, improved system throughput, much less I/O traffic and deliver a sustained high I/O performance
- CTFS improves system throughput from 24% to 37% compared to FFS-async when the cluster size varies up to 64