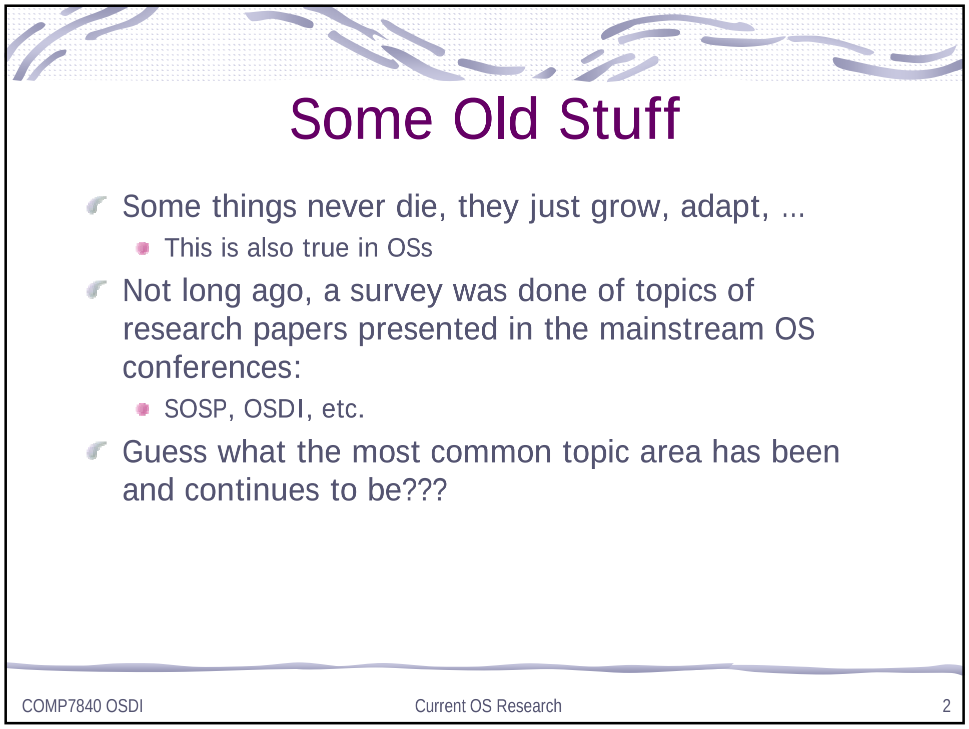
Current Topics in OS Research

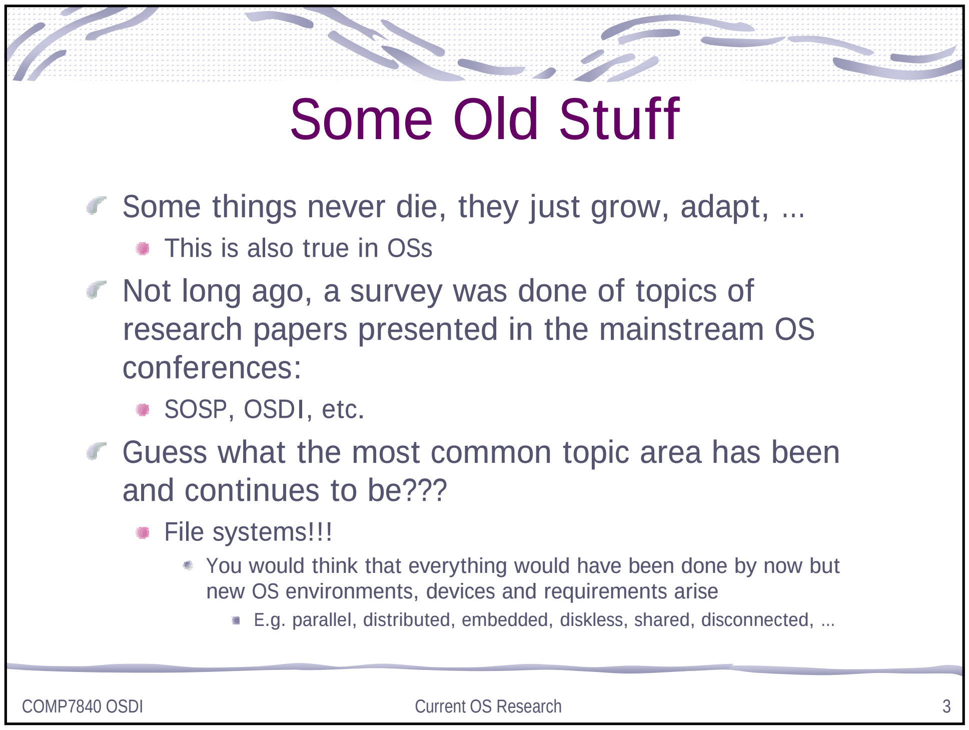
So, what's hot?

- Operating systems have been around for a long time in many forms for different types of devices
- It is normally general purposes OSs that are studied in undergraduate OS courses
  - This barely scratches the surface
    - Excluding real time, embedded, distributed, parallel, etc. OSs
  - But it does touch most of the key concepts covered in all types of OSs
    - Scheduling, memory management, device control, file systems, etc.
- In this part of the course, we will try to touch on some of the current research issues in OSs
  - Either stuff you haven't seen before or variants on what you have



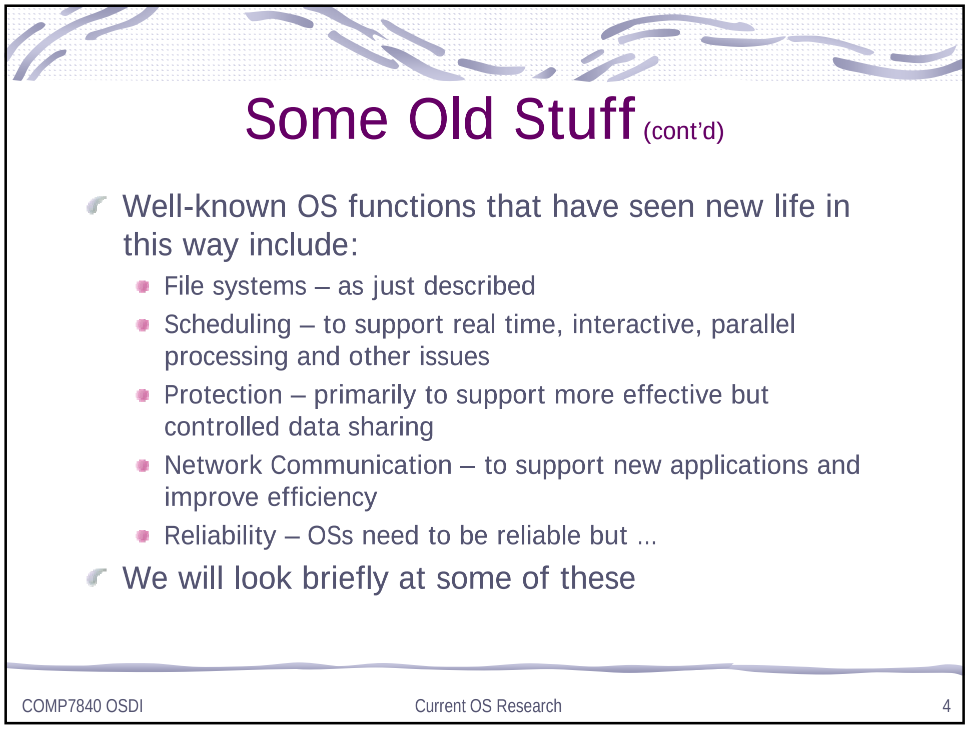
Some Old Stuff

- Some things never die, they just grow, adapt, ...
  - This is also true in OSs
- Not long ago, a survey was done of topics of research papers presented in the mainstream OS conferences:
  - SOSP, OSDI, etc.
- Guess what the most common topic area has been and continues to be???



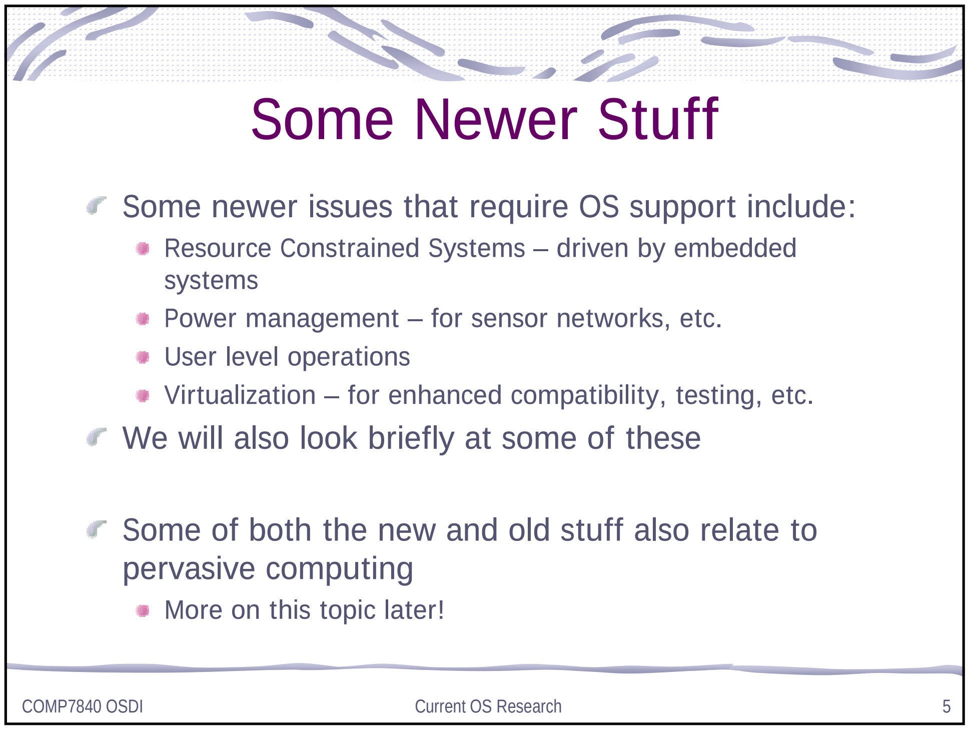
Some Old Stuff

- Some things never die, they just grow, adapt, ...
  - This is also true in OSs
- Not long ago, a survey was done of topics of research papers presented in the mainstream OS conferences:
  - SOSP, OSDI, etc.
- Guess what the most common topic area has been and continues to be???
  - File systems!!!
    - You would think that everything would have been done by now but new OS environments, devices and requirements arise
      - E.g. parallel, distributed, embedded, diskless, shared, disconnected, ...



Some Old Stuff (cont'd)

- Well-known OS functions that have seen new life in this way include:
  - File systems – as just described
  - Scheduling – to support real time, interactive, parallel processing and other issues
  - Protection – primarily to support more effective but controlled data sharing
  - Network Communication – to support new applications and improve efficiency
  - Reliability – OSs need to be reliable but ...
- We will look briefly at some of these



Some Newer Stuff

- Some newer issues that require OS support include:
  - Resource Constrained Systems – driven by embedded systems
  - Power management – for sensor networks, etc.
  - User level operations
  - Virtualization – for enhanced compatibility, testing, etc.
- We will also look briefly at some of these
- Some of both the new and old stuff also relate to pervasive computing
  - More on this topic later!

Some Recent Issues in File Systems

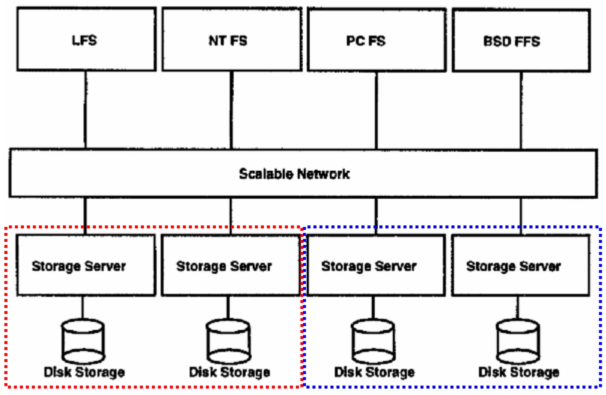
Distributed File Systems

- Probably the most common current topic in file systems is support for distributed file (and block) storage services
  - Driven by the growing prevalence of larger and new types of distributed computing environments
- Everyone knows about NFS but it has several limitations
- We'll look at a couple of other systems:
  - Petal & Frangipani: a 2-layer DFS system
  - xFS: a server-less network file system

Petal: Distributed Virtual Disks

- A distributed storage system that provides a virtual *disk* abstraction separated from the physical resources
- The virtual disk is globally accessible to all clients on the network
- Implemented on a cluster of network-connected servers that cooperate to manage a pool of physical disks
- Advantages
  - transparent reconfiguration and expandability
  - load and capacity balancing
  - lower-level service (block level) than a DFS
    - Flexible and good performance

Petal



Virtual to Physical Translation

- <virtual disk id, offset> -> <server, disk, offset>
- Translation must be valid in the presence of failure and subsequent recoveries
- Three data structures: virtual disk directory, global map, and physical map
- The virtual disk directory and global map are globally replicated and kept consistent
- One level of indirection (virtual disk to global map) is necessary to allow transparent reconfiguration
- Physical map is local to each server

Virtual to Physical Translation (2)

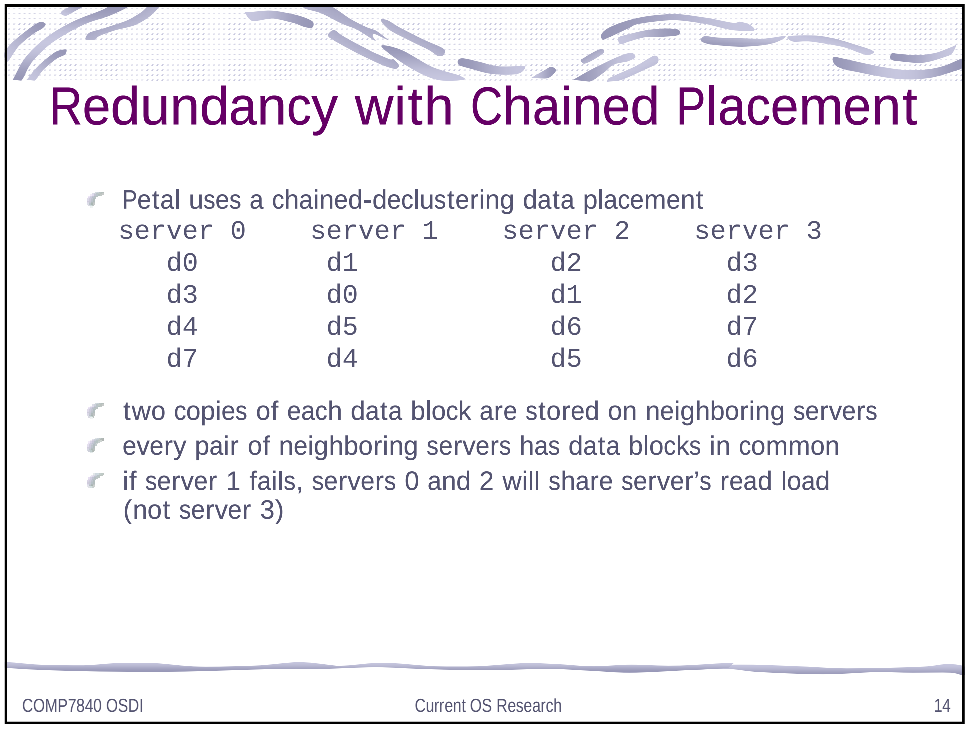
1. The virtual disk directory translates the virtual disk identifier into a global map identifier
2. The global map determines the server responsible for translating the given offset (could be more than one due to replication)
3. The physical map at a specific server translates the global map identifier and the offset to a physical disk and an offset within that disk

Virtual Disk Reconfiguration

- Needed when a new server is added or the redundancy scheme is changed
- Steps to perform at once (not incrementally):
  - create a new global map
  - change all virtual disk directories to point to the new global map
  - redistribute data to the servers according to the translation specified in the new global map
- The challenge is to perform reconfiguration incrementally
  - So we don't shut down the whole system for long periods of time

Incremental Reconfiguration

- First two steps as before; step 3 done in background starting with the translations in the most recent epoch that have not yet been moved
- Old global map is used to perform translations which are not found in the new global map
- A write request will always access only the new global map
- By moving data starting with the most recent epoch we avoid the situation where a read request returns data from an older epoch than the one requested by the client
- Limitation: the mapping of the entire virtual disk must be changed before any data is moved

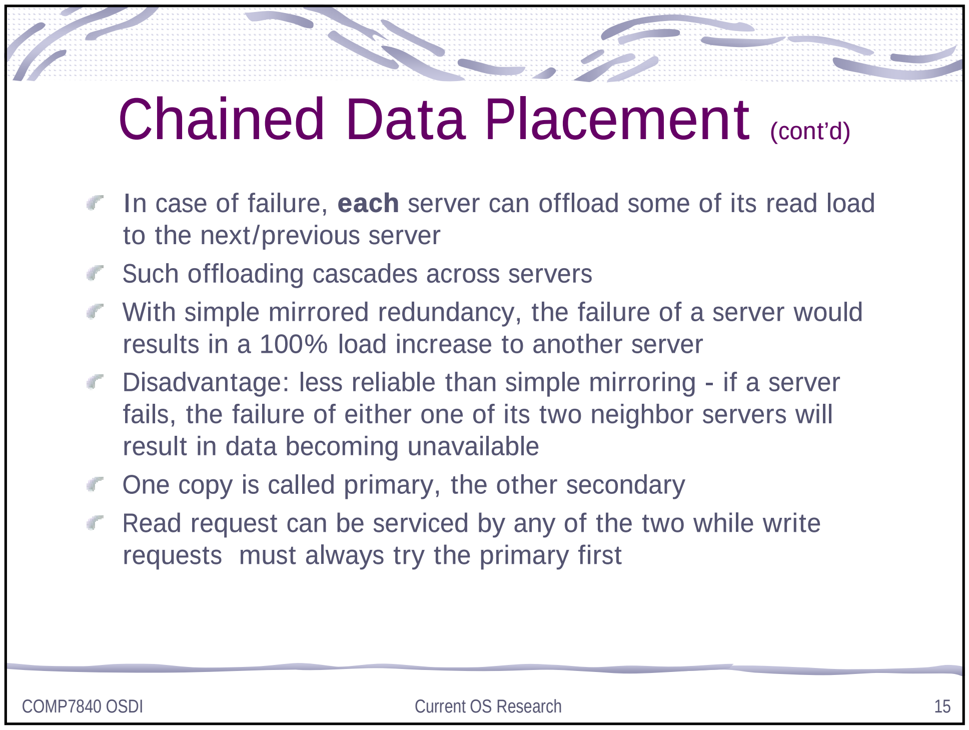


Redundancy with Chained Placement

- Petal uses a chained-declustering data placement

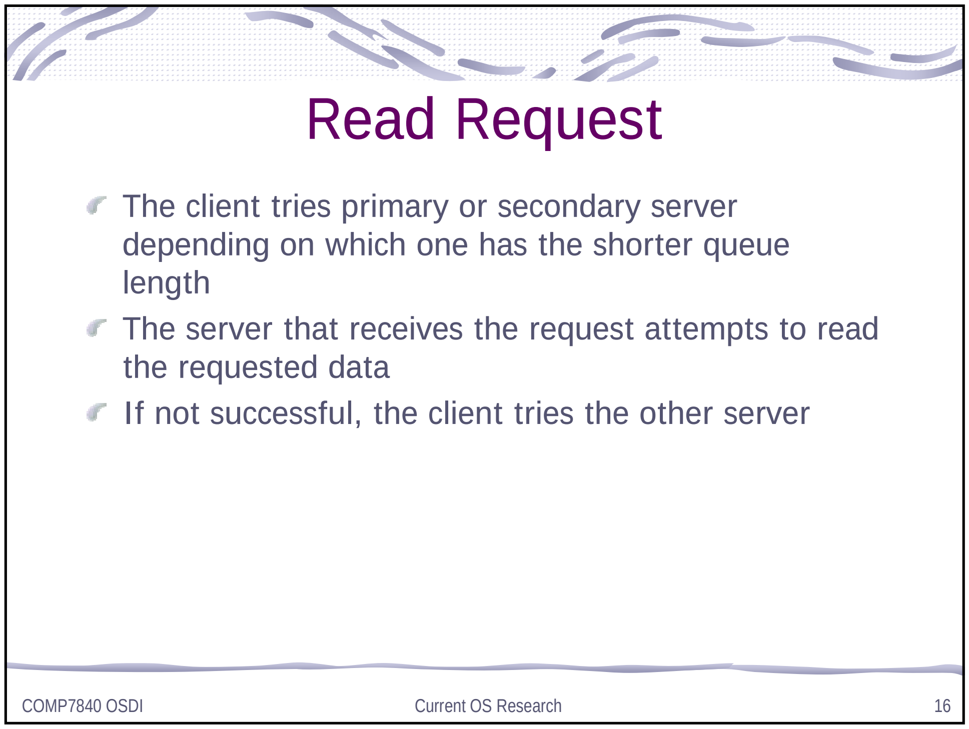
server 0	server 1	server 2	server 3
d0	d1	d2	d3
d3	d0	d1	d2
d4	d5	d6	d7
d7	d4	d5	d6

- two copies of each data block are stored on neighboring servers
- every pair of neighboring servers has data blocks in common
- if server 1 fails, servers 0 and 2 will share server's read load (not server 3)



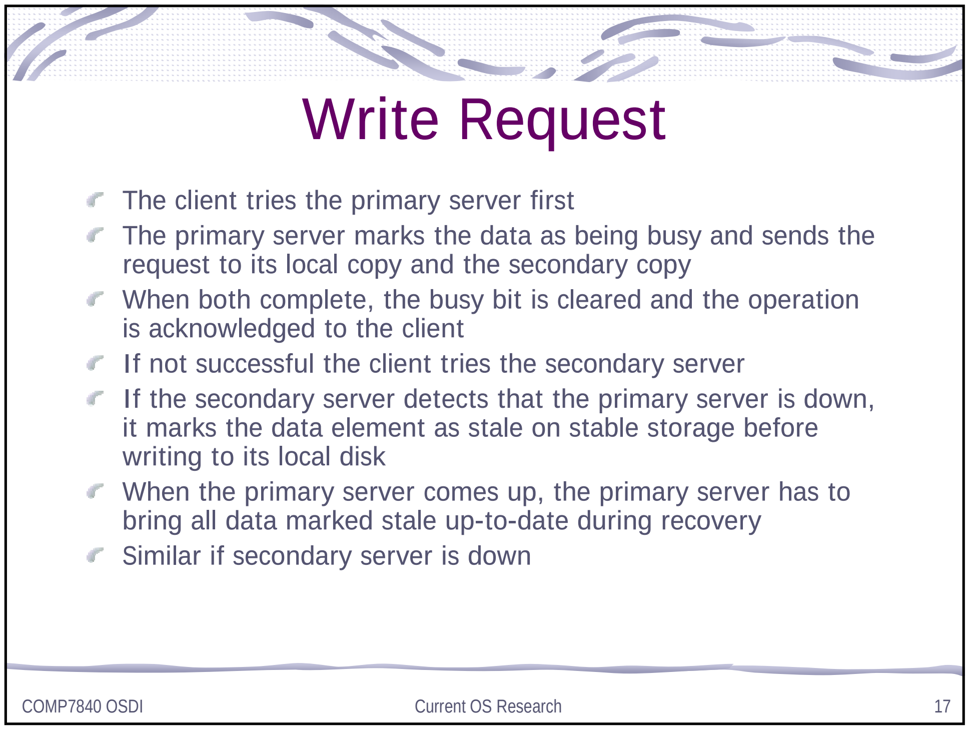
Chained Data Placement (cont'd)

- In case of failure, **each** server can offload some of its read load to the next/previous server
- Such offloading cascades across servers
- With simple mirrored redundancy, the failure of a server would results in a 100% load increase to another server
- Disadvantage: less reliable than simple mirroring - if a server fails, the failure of either one of its two neighbor servers will result in data becoming unavailable
- One copy is called primary, the other secondary
- Read request can be serviced by any of the two while write requests must always try the primary first



Read Request

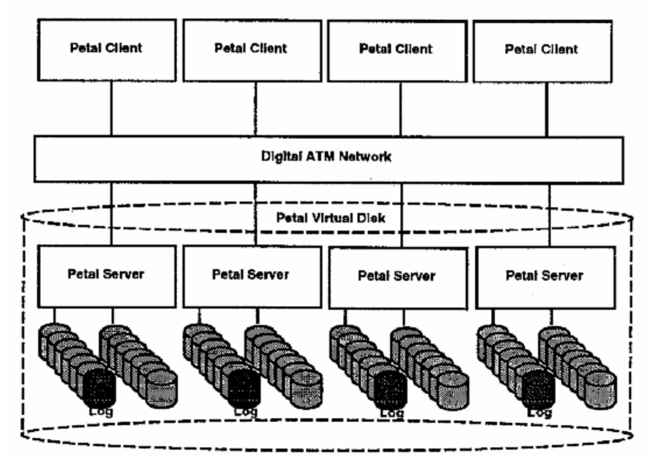
- The client tries primary or secondary server depending on which one has the shorter queue length
- The server that receives the request attempts to read the requested data
- If not successful, the client tries the other server



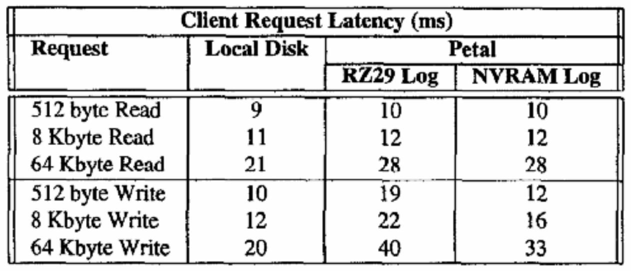
Write Request

- The client tries the primary server first
- The primary server marks the data as being busy and sends the request to its local copy and the secondary copy
- When both complete, the busy bit is cleared and the operation is acknowledged to the client
- If not successful the client tries the secondary server
- If the secondary server detects that the primary server is down, it marks the data element as stale on stable storage before writing to its local disk
- When the primary server comes up, the primary server has to bring all data marked stale up-to-date during recovery
- Similar if secondary server is down

Petal Prototype



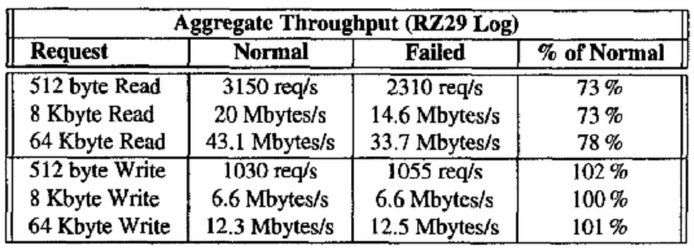
Petal Performance - Latency



Client Request Latency (ms)			
Request	Local Disk	Petal	
		RZ29 Log	NVRAM Log
512 byte Read	9	10	10
8 Kbyte Read	11	12	12
64 Kbyte Read	21	28	28
512 byte Write	10	19	12
8 Kbyte Write	12	22	16
64 Kbyte Write	20	40	33

Table 1: Latency of a Chained-Declustered Virtual Disk

Petal Performance - Throughput



Aggregate Throughput (RZ29 Log)			
Request	**Normal**	**Failed**	**% of Normal**
512 byte Read	3150 req/s	2310 req/s	73 %
8 Kbyte Read	20 Mbytes/s	14.6 Mbytes/s	73 %
64 Kbyte Read	43.1 Mbytes/s	33.7 Mbytes/s	78 %
512 byte Write	1030 req/s	1055 req/s	102 %
8 Kbyte Write	6.6 Mbytes/s	6.6 Mbytes/s	100 %
64 Kbyte Write	12.3 Mbytes/s	12.5 Mbytes/s	101 %

Table 2: Normal and Failed Throughput of a Chained-Declustered Virtual Disk

Petal Performance - Scalability

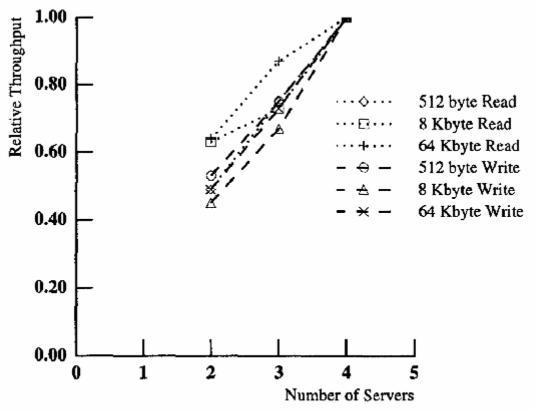
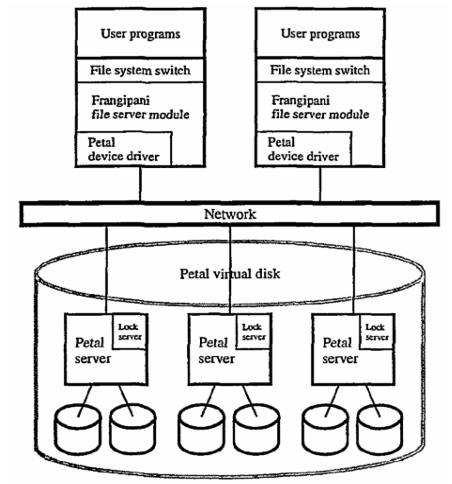


Figure 7: Scaling with Increased Servers

Frangipani

- Petal provides disk interface -> you still need a file system
- Frangipani (runs over Petal)
  - All users have a consistent view of the file namespace
  - Servers can be added without changing configuration of existing servers
  - Tolerates and recovers from machine, network, and disk failures
- Is very simple internally: a set of cooperating machines using a common store that synchronize access to that store with locks
  - Petal takes much of the complexity out of Frangipani
  - Cannot use disk location in placing data

Frangipani Structure



Frangipani: Disk Layout

- A Frangipani file system uses only 1 Petal virtual disk
- Petal provides 2^{64} bytes of “virtual” disk space
  - Commits real disk space when actually used, usually sparse
- Frangipani breaks disk into regions
  - 1st region stores configuration parameters
  - 2nd region stores logs – each Frangipani server uses a portion of this region for its log. Can have up to 256 logs.
  - 3rd region holds allocation bitmaps, describing which blocks in the remaining regions are free. Each server locks a different portion.
  - 4th region holds inodes
    - Mapping between bits in allocation bitmap and inodes is fixed
    - Each server only allocate inodes from its region
  - 5th region holds small data blocks
  - Remainder of Petal disk holds large data blocks

Frangipani: File Structure

- First 16 blocks (64 KB) of a file are stored in small blocks
- If file becomes larger, store the rest in a 1 TB large block

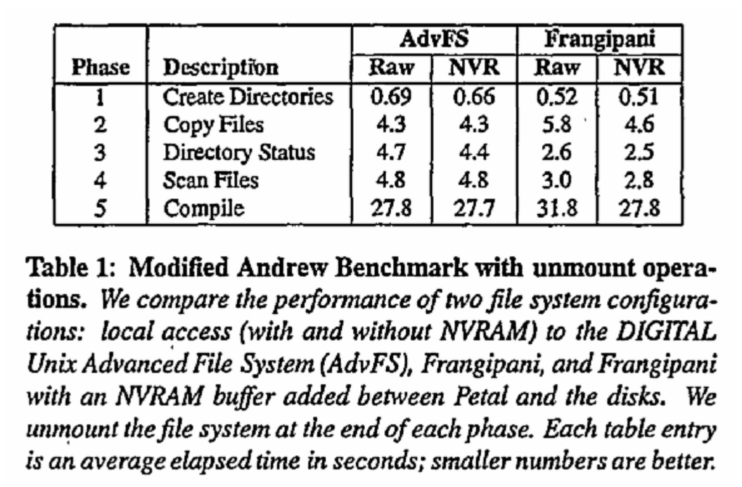
Frangipani: Dealing with Failures

- Write-ahead redo logging of metadata
- Each server has its own private log
- Only after a log record is written to Petal does the server modify the actual metadata in its permanent location(s)
- If a server crashes, the system detects the failure and another server uses the log to recover
  - Because the log is on Petal, any server can get to it.

Frangipani: Coherency

- Have a lock for each log segment, allocation bitmap segment, and each file
- Multiple readers/single writer locks
- Distributed lock service
  - Similar issues to Petal
    - How to keep state coherent
    - How to detect/recover from failure
- How to take back locks from a failed server?
- What about adding/removing servers?

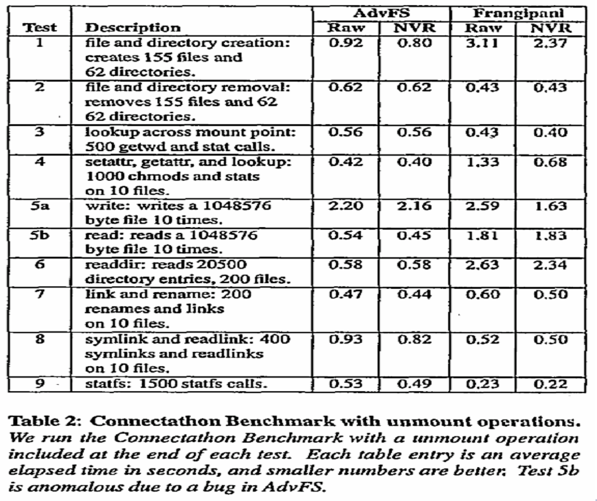
Frangipani: Performance



Phase	Description	AdvFS		Frangipani	
		Raw	NVR	Raw	NVR
1	Create Directories	0.69	0.66	0.52	0.51
2	Copy Files	4.3	4.3	5.8	4.6
3	Directory Status	4.7	4.4	2.6	2.5
4	Scan Files	4.8	4.8	3.0	2.8
5	Compile	27.8	27.7	31.8	27.8

Table 1: Modified Andrew Benchmark with unmount operations. *We compare the performance of two file system configurations: local access (with and without NVRAM) to the DIGITAL Unix Advanced File System (AdvFS), Frangipani, and Frangipani with an NVRAM buffer added between Petal and the disks. We unmount the file system at the end of each phase. Each table entry is an average elapsed time in seconds; smaller numbers are better.*

Frangipani: Performance



Test	Description	AdvFS		Frangipani	
		Raw	NVR	Raw	NVR
1	file and directory creation: creates 155 files and 62 directories.	0.92	0.80	3.11	2.37
2	file and directory removal: removes 155 files and 62 62 directories.	0.62	0.62	0.43	0.43
3	lookup across mount point: 500 getwd and stat calls.	0.56	0.56	0.43	0.40
4	setattr, getattr, and lookup: 1000 chmods and stats on 10 files.	0.42	0.40	1.33	0.68
5a	write: writes a 1048576 byte file 10 times.	2.20	2.16	2.59	1.63
5b	read: reads a 1048576 byte file 10 times.	0.54	0.45	1.81	1.83
6	readdir: reads 20500 directory entries, 200 files.	0.58	0.58	2.63	2.34
7	link and rename: 200 renames and links on 10 files.	0.47	0.44	0.60	0.50
8	symlink and readlink: 400 symlinks and readlinks on 10 files.	0.93	0.82	0.52	0.50
9	statfs: 1500 statfs calls.	0.53	0.49	0.23	0.22

Table 2: Connectathon Benchmark with unmount operations. *We run the Connectathon Benchmark with a unmount operation included at the end of each test. Each table entry is an average elapsed time in seconds, and smaller numbers are better. Test 5b is anomalous due to a bug in AdvFS.*

Frangipani: Scalability

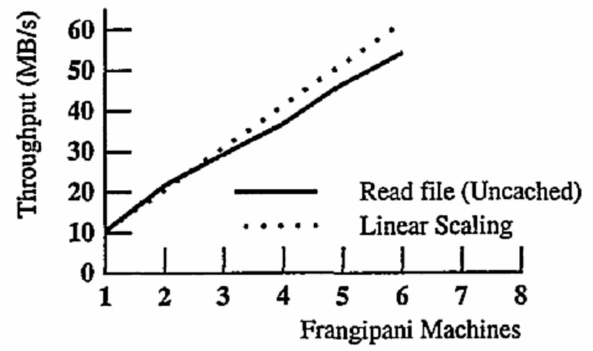


Figure 6: Frangipani Scaling on Uncached Read. *Several Frangipani servers simultaneously read the same set of files. The dotted line shows the linear speedup curve for comparison.*

Frangipani: Scalability

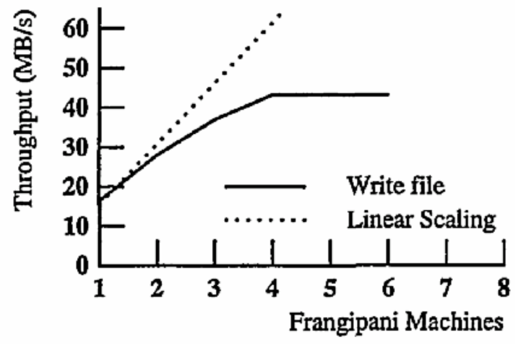
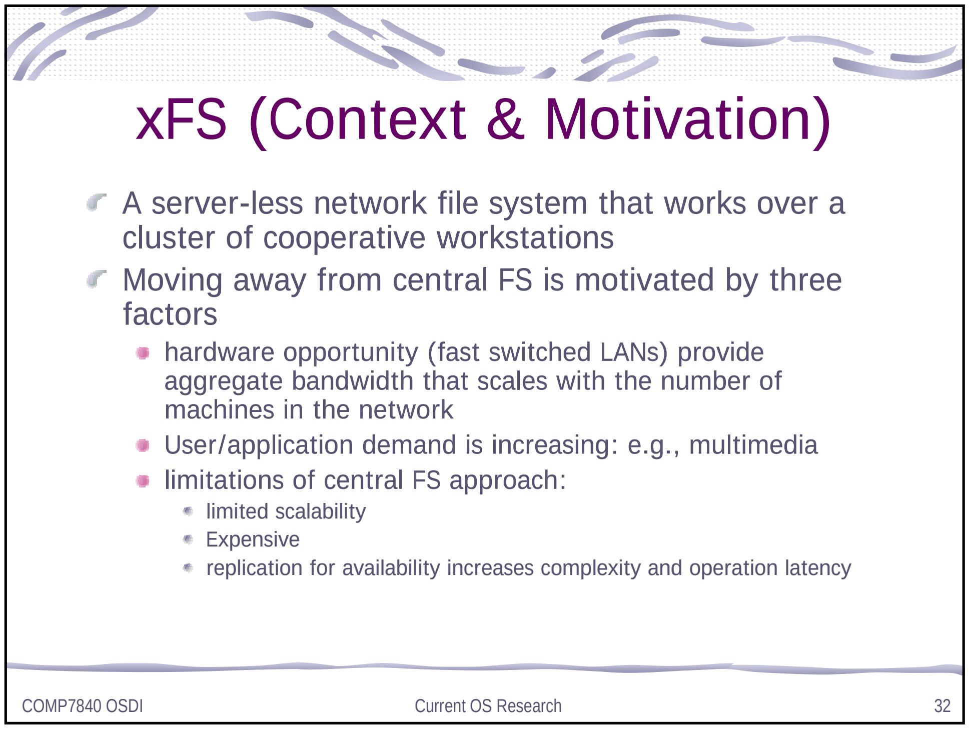
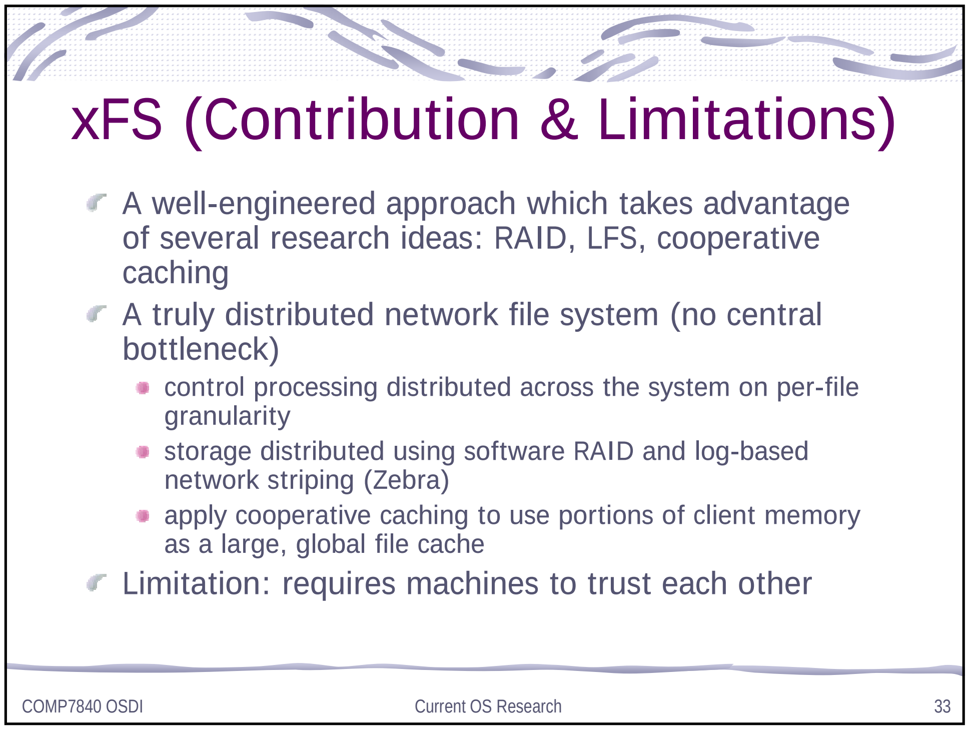


Figure 7: Frangipani Scaling on Write. *Each Frangipani server writes a large private file. The dotted line shows the linear speedup curve for comparison. Performance tapers off early because the ATM links to the Petal servers become saturated.*



xFS (Context & Motivation)

- A server-less network file system that works over a cluster of cooperative workstations
- Moving away from central FS is motivated by three factors
  - hardware opportunity (fast switched LANs) provide aggregate bandwidth that scales with the number of machines in the network
  - User/application demand is increasing: e.g., multimedia
  - limitations of central FS approach:
    - limited scalability
    - Expensive
    - replication for availability increases complexity and operation latency



xFS (Contribution & Limitations)

- A well-engineered approach which takes advantage of several research ideas: RAID, LFS, cooperative caching
- A truly distributed network file system (no central bottleneck)
  - control processing distributed across the system on per-file granularity
  - storage distributed using software RAID and log-based network striping (Zebra)
  - apply cooperative caching to use portions of client memory as a large, global file cache
- Limitation: requires machines to trust each other

RAID in xFS

- RAID partitions a stripe of data into N-1 data blocks and a parity block (the exclusive-OR of the bits of data blocks)
- Data and parity blocks are stored on different storage servers
- Provides both high bandwidth and fault tolerance
- Traditional RAID drawbacks:
  - multiple accesses for small writes
  - hardware RAID adds expense (special hardware to compute parity)

LFS in xFS

- High-performance writes: buffer writes in memory to write them to disk in large, contiguous, fixed-size groups called log segments
- Writes are always appended as logs
- imap to locate i-nodes: stored in memory and periodically check pointed to disk
- Simple recovery procedure: get the last checkpoint and then roll forward reading the later segments in the log and update imap and i-nodes
- Free disk management through log cleaner: coalesces old, partially empty segments into a smaller number of full segments -> cleaning overhead can be large sometime

Zebra

- Combines LFS and RAID: LFS's large writes make writes to the network RAID more efficient
- Implements RAID in software
- Log-based striping:
  - log segment split into log fragments which are striped over the storage servers
  - parity fragment computation is local (no network access)
- Deltas stored in the log encapsulate modifications to file system state that must be performed atomically - used for recovery

Metadata and Data Distribution

- A centralized FS stores all data blocks on its local disks
  - manages location of metadata
  - maintains a central cache of data blocks in its memory
  - manages cache consistency metadata that lists which clients in the system are caching each block (not NFS)
- But, there is no central location in xFS

xFS: Metadata & Data Distribution

- Stores data on storage servers
- Splits metadata management among multiple managers that can dynamically alter the mapping from a file to its manager
- Uses cooperative caching that forwards data among client caches under the control of the managers
- The key design challenge: how to locate data and metadata in such a completely distributed system

xFS: Performance

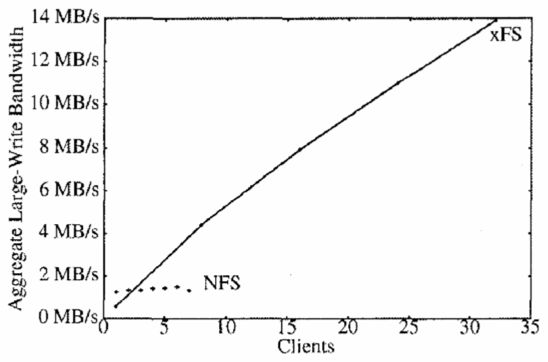


Figure 9. Aggregate disk write bandwidth. The x axis indicates the number of clients simultaneously writing private 10 MB files, and the y axis indicates the total throughput across all of the active clients. xFS used four groups of eight storage servers and 32 managers. NFS's peak throughput is 1.5 MB/s with 6 clients; xFS's is 13.9 MB/s with 32 clients.

xFS: Performance

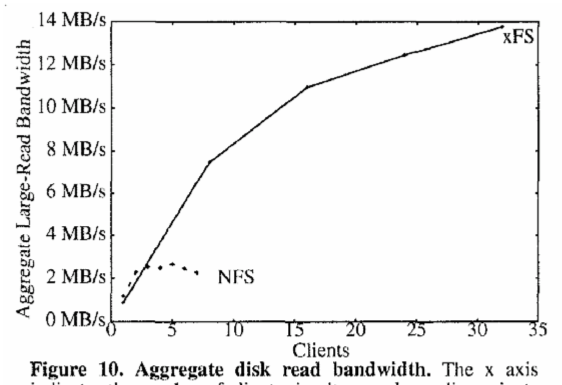


Figure 10. Aggregate disk read bandwidth. The x axis indicates the number of clients simultaneously reading private 10 MB files and the y axis indicates the total throughput across all active clients. xFS used four groups of eight storage servers and 32 managers. NFS's peak throughput is 2.7 MB/s with 5 clients; xFS's is 13.8 MB/s with 32 clients.