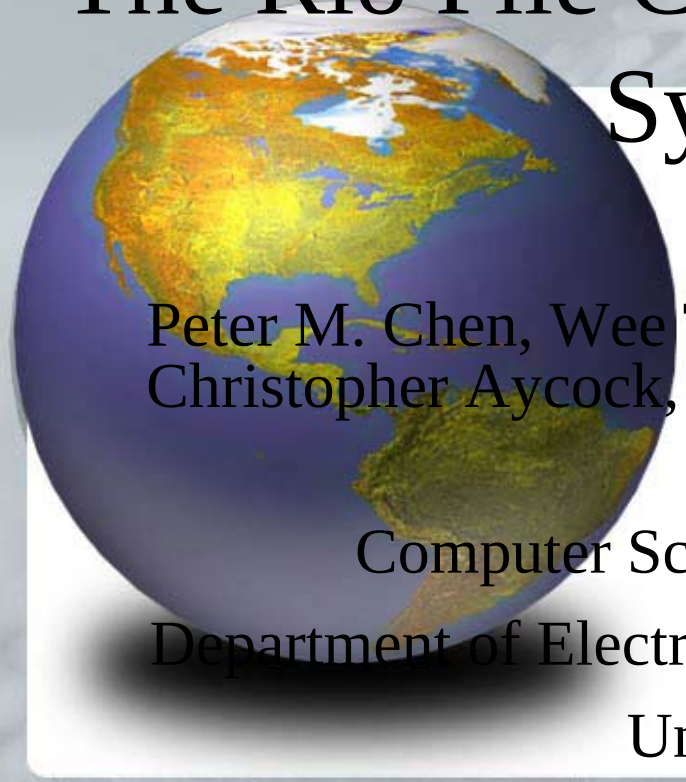
The Rio File Cache: Surviving Operating System Crashes



Peter M. Chen, Wee Teck Ng, Subhachandra Chandra,
Christopher Aycock, Gurushankar Rajamani, David Lowell

Computer Science and Engineering Division

Department of Electrical Engineering and Computer Science

University of Michigan

1996 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)

Outline

- **Introduction**
- **Design and Implementation**
  - **Protection**
  - **Warm Reboot**
  - **Effects on File System Design**
- **Reliability**
- **Performance**
- **Architeture Support**
- **Conclusion**

Introduction – 1/2

- Memory's vulnerability to power outages is easy to understand and fix.
- Memory's vulnerability to OS crashes is more challenging.

Introduction – 2/2

- A tradeoff between performance and reliability.
  - Asynchronously writing data to disk -- a greater degree of overlap between CPU time and I/O time.
  - delaying some writes to disk. This delay is often set to 30 seconds.
  - maximum performance : a pure write-back scheme -- data is written to disk only when the memory is full.

Design and Implementation : Protection – 1/3

- Access interface is the reason most people view battery-backed memory more vulnerable than disk.
- The interface used to access disk is complex.
- In contrast, the interface used to access memory is simple. It is hence relatively easy for many simple software errors (EX. de-referencing an uninitialized pointer.)

Design and Implementation : Protection – 2/3

- Unified Buffer Cache(regular files) is sometimes not mapped into viutual page –”dynamically” conserve TLB slot.
- Turn off the write-permission bits in the page table to do protection.
- File cache procedure can check corruption after every writing and implement atomic write.

Design and Implementation : Protection – 3/3

- However, some processor can’t disable bypassing TLB access, and we must insert “code patching” in the kernel.

- In every write to physical address, inert code will make sure it’s not in file cache or the file cache has registered this address as writable. (20%~50% slower).

Design and Implementation : Warm Reboot

- Two issues arise during a warm reboot:
  - what additional data the system maintains
  - when the reboot process restores the file cache contents.

- Additional data–which we call registry (file ID, physical memory address, file size) .

- During reboot, before OS and file system initialization, we dump all physical memory to a partition, and restore dirty file cache to disk by using registry.

Design and Implementation : Effects on File System Design

- We disable buffer cache writes by turning most bwrite and bawrite calls to bdwrite.
- we modify sync and fsync calls to return immediately
- Atomicity
  - When system write data into file cache, it first copies the contents to a shadow page and changes the registry entry to point to the shadow.
  - When writing finishes, it atomically points the registry entry back to the original buffer.

Reliability :Two Kinds of Corruption

- *direct* corruption : detected by checksum
  - a series of events eventually causes a non-I/O procedure to accidentally write to file data.
- *indirect* corruption : detected by App level check
  - a series of events eventually causes a procedure to call an I/O procedure with the wrong parameters.
- However, mechanism mentioned before can’t protect against *indirect* corruption.

Reliability : Detecting Indirect Corruption

- We use app ***memTest*** to detect indirect corruption
- ***memTest*** record the stream of file and directory creations, deletions, reads, and writes, and save this status file in the network.
- After system crash and reboot, we can use this stream to reconstruct the correct content to compare with file cache.

Fault Type	Disk-Based	Rio without Protection	Rio with Protection
kernel text	2	1	
kernel heap			
kernel stack		1	1
destination reg.			
source reg.	2		
delete branch	1	1	1
delete random inst.	1		
initialization			1
pointer		1	
allocation			
copy overrun		4	
off-by-one	1	2	1
synchronization			
Total	**7 of 650 (1.1%)**	**10 of 650 (1.5%)**	**4 of 650 (0.6%)**

	Data Permanent	cp+rm (seconds)
Memory File System	never	21 (15+6)
UFS with delayed data and metadata	after 0-30 seconds, asynchronous	81 (76+5)
AdvFS (log metadata updates)	after 0-30 seconds, asynchronous	125 (110+15)
UFS	data after 64 KB, asynchronous metadata synchronous	332 (245+87)
UFS with write-through after each close	after close, synchronous	394 (274+120)
UFS with write-through after each write	after write, synchronous	539 (419+120)
Rio without protection	**after write, synchronous**	**24 (18+6)**
Rio with protection	**after write, synchronous**	**25 (18+7)**

	cp+rm (seconds)	Sdet (5 scripts) (seconds)
Memory File System	21 (15+6)	43
UFS with delayed data and metadata	81 (76+5)	47
AdvFS (log metadata updates)	125 (110+15)	132
UFS	332 (245+87)	401
UFS with write-through after each close	394 (274+120)	699
UFS with write-through after each write	539 (419+120)	910
Rio without protection	**24 (18+6)**	**42**
Rio with protection	**25 (18+7)**	**42**

Architecture Support

- First, hardware should provide the ability to force all accesses through the TLB.

- System should be able to be reset and rebooted without erasing the contents of memory or CPU caches containing memory data.

Conclusion

- Future work : such as writing to disk during idle periods may improve system responsiveness, and we plan to experiment with this in the future.

- We could perform a more wide variety of fault-injection experiment in the future.