

Integrating Reliable Memory in Databases

Wee Teck Ng, Peter M. Chen

**Computer Science and Engineering Division
Electrical Engineering and Computer Science
University of Michigan**

Objectives

Introduce the Rio file cache

software-based reliable memory

Examine three different ways of using reliable memory and their implications

I/O interface: non-persistent database buffer cache

memory interface: persistent database buffer cache

protected memory interface: protected persistent database buffer cache

Show quantitatively that it is safe to mmap reliable memory directly into database address space

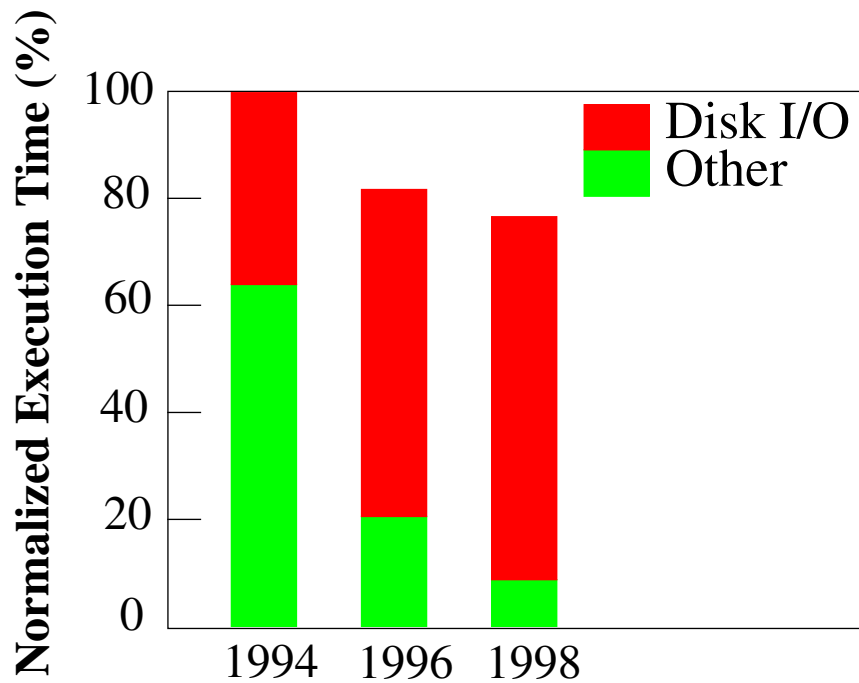
simplify database design

significant performance advantage

Reliable Memory

**Reliable memory = main memory as safe as disk
from operating system crashes**

**Reliable memory is important for current and future
database systems [Rosenblum95]**

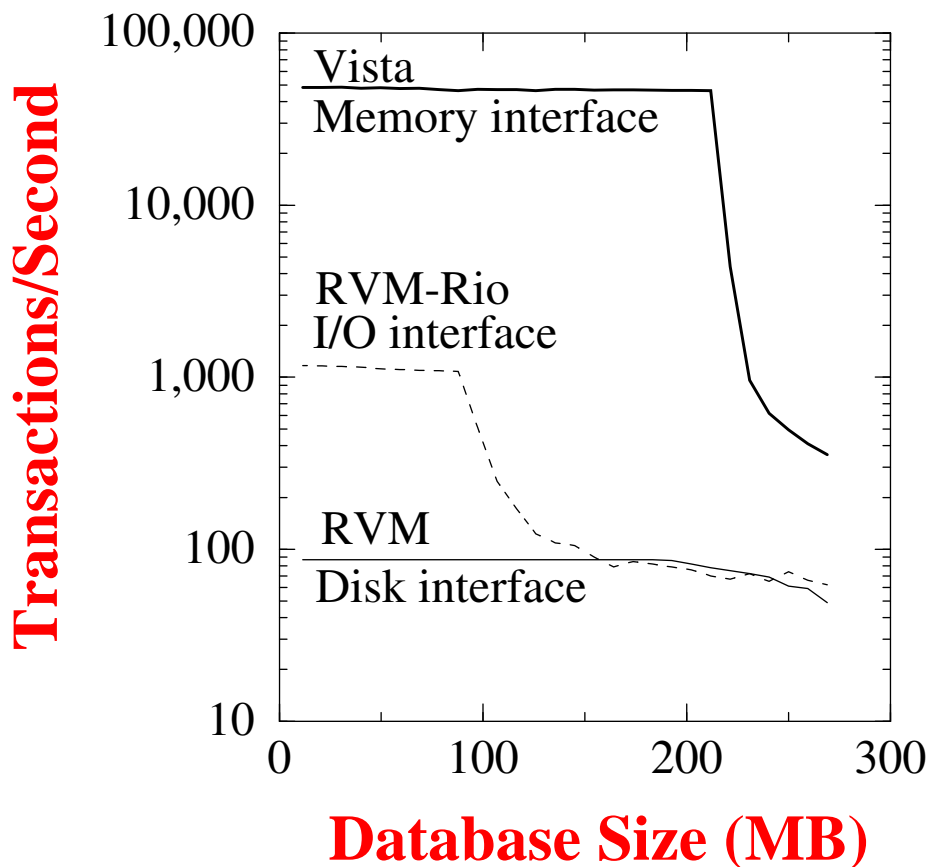


Benefits of Reliable Memory

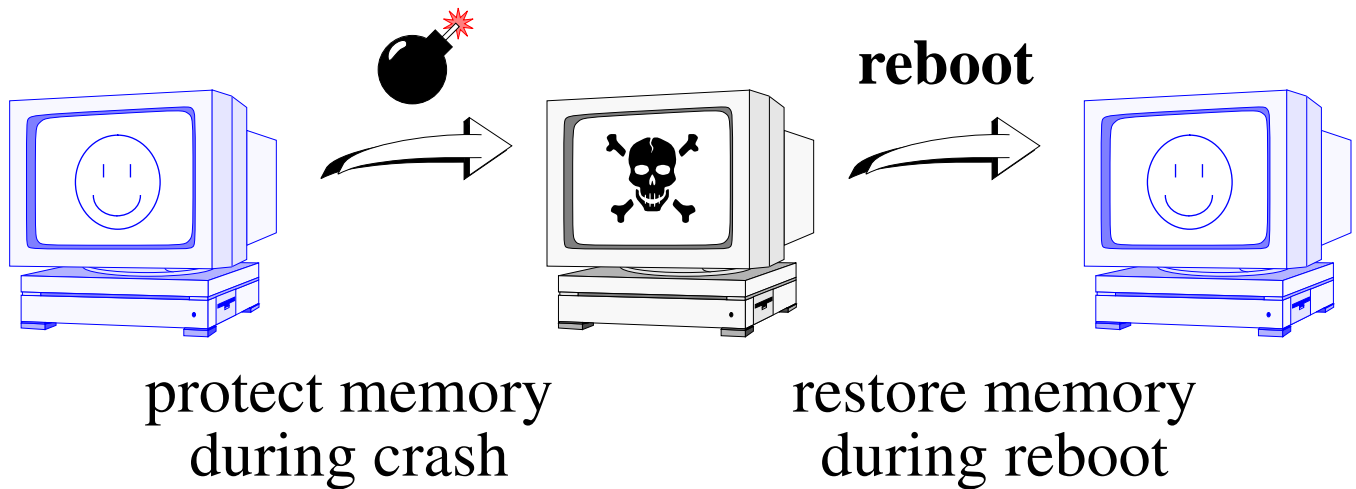
Simplify database design

- fast recovery (no redo log, checkpt, warm cache)
- direct control of buffer cache (optimized for DB)
- simpler and smaller code (less bugs)

Significant performance improvements



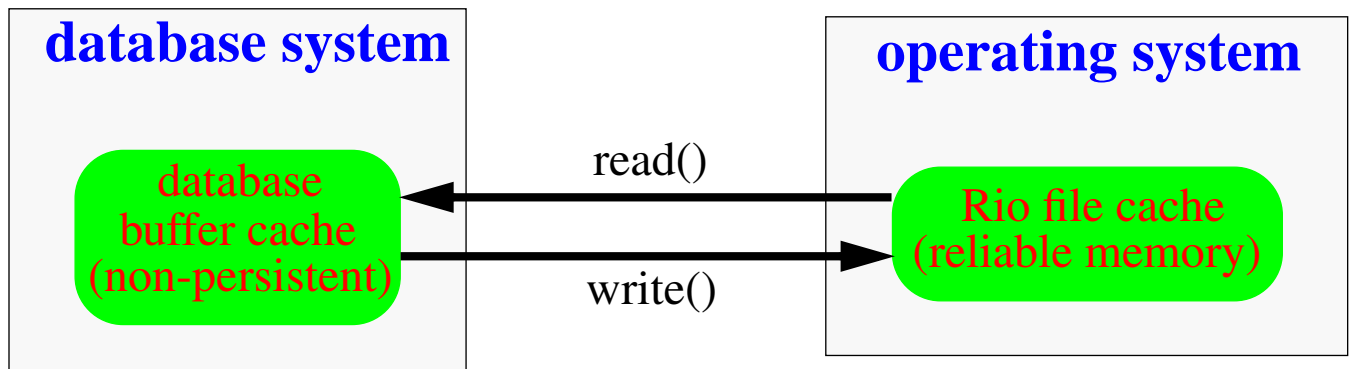
Rio: A Reliable File Cache



Remove reliability-induced writes to disk (sync, fsync, bwrite, bawrite)

Makes memory safe from operating system crashes, requires no hardware

I/O Interface to Reliable Memory



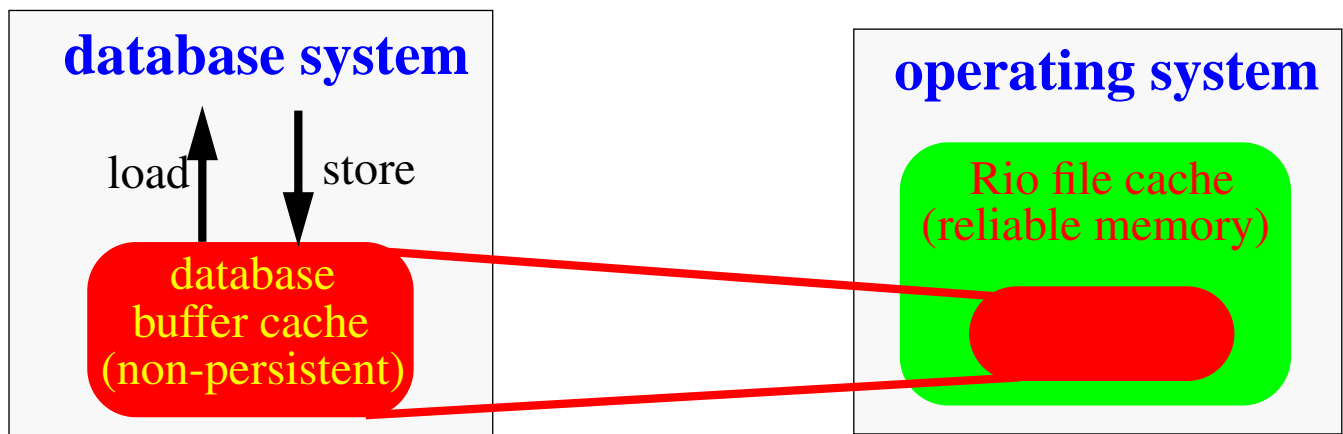
Benefits

- + performance improvement without any change to application code
- + as safe as standard database from operating system crashes

Drawbacks

- performance (we can do better)
- double buffering

Memory Interface to Reliable Memory



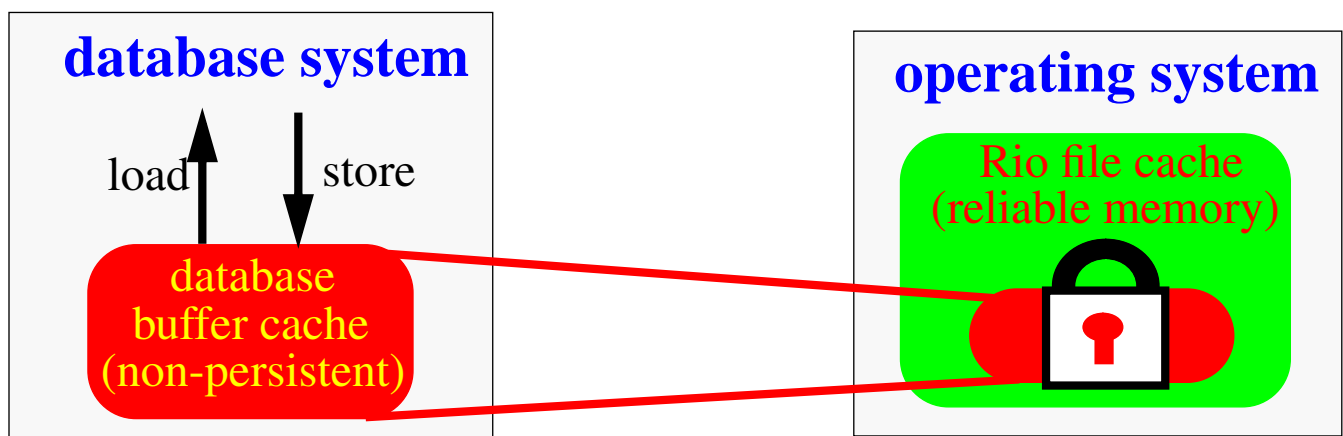
Benefits

- + significant performance improvements
- + simplify database design
- + safe from operating system *and* database failures

Drawbacks

- require some changes in database design
- increase database's vulnerability to software errors

Protected Memory Interface



Benefits

- + same as memory interface
- + minimize database's vulnerability to software errors

Drawbacks

- affect performance due to protection overheads
- does it really help?

Fault Models

Goal is wide variety and realism [Sullivan91]

Fault Type	Example of Programming Error
destination register	numFreePages = count(freePageHeadPtr)
source register	numPages = physicalMemorySize /pageSize
delete branch	if while (...) {body}
delete random inst.	for (i=0; i<10; i++, j++) {body}
initialization	function () {int i = 0 ; ...}
pointer	ptr = ptr->next => next ;
allocation	ptr = malloc(); use ptr; use ptr; free(ptr) ;
copy overrun	for (i=0; i < sizeUsed ; i++) {a[i] = b[i]};
off-by-one	for (i=0; i < <= size; i++)
synchronization	getWriteLock ; write(); freeWriteLock ;
memory leak	{code} free(ptr) ; {more code}
interface error	insert(buf Temp , index);

Reliability Results

Fault Type	I/O Interface		
kernel text	1		
kernel heap			
kernel stack			
destination register	4		
source register	2		
delete branch	1		
delete random inst.	2		
initialization			
pointer			
allocation			
copy overrun			
off-by-one	5		
synchronization			
memory leak			
interface error	4		
Total	19 of 750 (2.5%)		

		Memory Interface	
		1	
		5	
		2	
		1	
		2	
		1	
		5	
		3	
		20 of 750 (2.7%)	

			Protected Memory I/F
			1
			5
			2
			2
			1
			3
			3
			17 of 750 (2.3%)

Conclusions

Reliable memory yields huge benefits, particularly when using a memory interface to it (persistent buffer cache)

Memory interface does not hurt reliability, so we should use it

Need to verify our findings on commercial database

More information available at

<http://www.eecs.umich.edu/~pmchen/>