EECS 262a
Advanced Topics in Computer Systems
Lecture 8

Transactional Flash & Rethink the Sync
September 29th, 2014

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

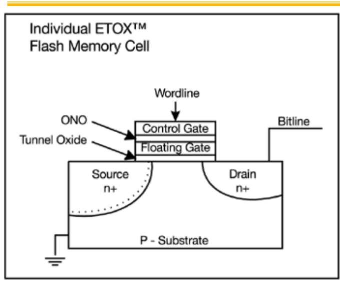
http://www.eecs.berkeley.edu/~kubitron/cs262

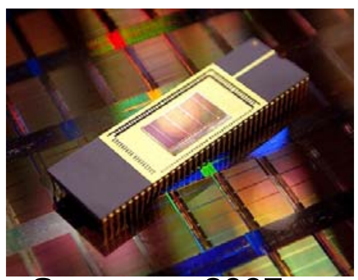
Today's Papers

- **Transactional Flash**
  Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou. Appears in Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI 2008).
- **Rethink the Sync**
  Edmund B. Nightingale, Kaushik Veeraraghavan, Peter M. Chen, and Jason Flinn. Appears in Proceedings of the 7th USENIX Conference on Operating Systems Design and Implementation (OSDI 2006).

- Thoughts?

FLASH Memory

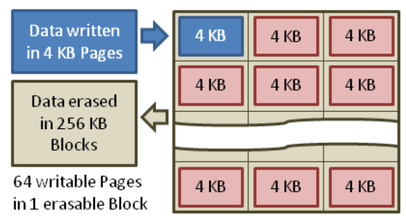




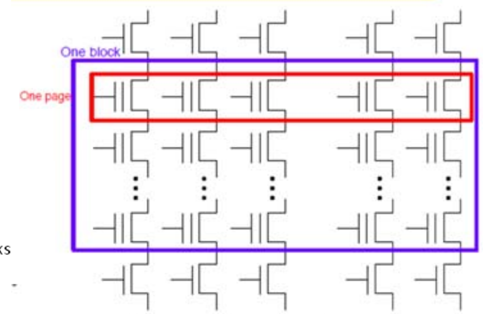
Samsung 2007:
16GB, NAND Flash

- **Like a normal transistor but:**
  - Has a floating gate that can hold charge
  - To write: raise or lower wordline high enough to cause charges to tunnel
  - To read: turn on wordline as if normal transistor
    - » presence of charge changes threshold and thus measured current
- **Two varieties:**
  - NAND: denser, must be read and written in blocks
  - NOR: much less dense, fast to read and write

Flash Memory (Con't)



Typical NAND Flash Pages and Blocks

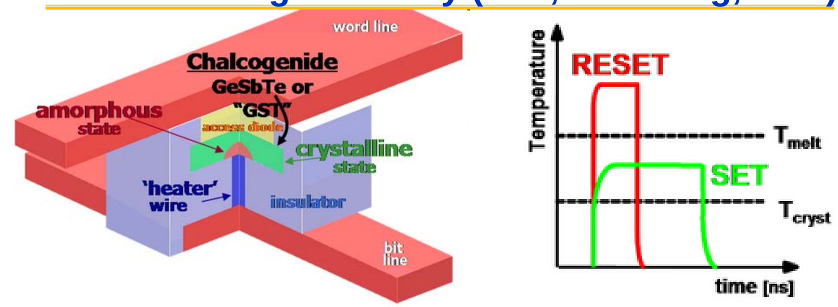


- **Data read and written in page-sized chunks (e.g. 4K)**
  - Cannot be addressed at byte level
  - Random access at block level for reads (no locality advantage)
  - Writing of new blocks handled in order (kinda like a log)
- **Before writing, must be *erased* (256K block at a time)**
  - Requires free-list management
  - CANNOT write over existing block (Copy-on-Write is normal case)

Flash Details

- Program/Erase (PE) Wear
  - Permanent damage to gate oxide at each flash cell
  - Caused by high program/erase voltages
  - Issues: trapped charges, premature leakage of charge
  - *Need to balance how frequently cells written: "Wear Leveling"*
- Flash Translation Layer (FTL)
  - Translates between Logical Block Addresses (at OS level) and Physical Flash Page Addresses
  - Manages the wear and erasure state of blocks and pages
  - Tracks which blocks are garbage but not erased
- Management Process (Firmware)
  - Keep freelist full, Manage mapping, Track wear state of pages
  - Copy good pages out of basically empty blocks before erasure
- Meta-Data per page:
  - ECC for data
  - Wear State
  - Other Stuff!: Capitalized on by this paper!

Phase Change memory (IBM, Samsung, Intel)



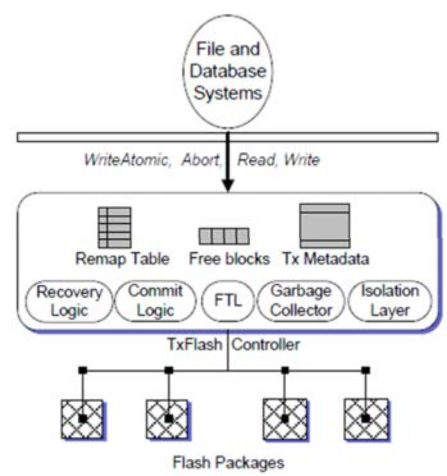
- Phase Change Memory (called PRAM or PCM)
  - Chalcogenide material can change from amorphous to crystalline state with application of heat
  - Two states have very different resistive properties
  - Similar to material used in CD-RW process
- Exciting alternative to FLASH
  - Higher speed
  - May be easy to integrate with CMOS processes

Goals of paper

- Provide a hardware Transactional model:
  - WriteAtomic(p1,p2,p3,…, p_n)
  - Interfering Reads not tracked
  - Transactions can be aborted before commited
- Provides:
  - Atomicity (All or nothing)
  - Isolation (Different transactions do not interfere)
  - Durability (After commit, data will survive crashes
- Target: file systems/databases
  - Provides a native implementation for durable log
  - However – provides its semantics without using a log (using linked metadata as the "log")
- Properties of Flash that is good for TxFlash:
  - Copy on Write is natural
  - Fast random reads (fragmentation of "log-based" system not a problem)
  - High Concurrency (lots of bandwidth could be exploited)

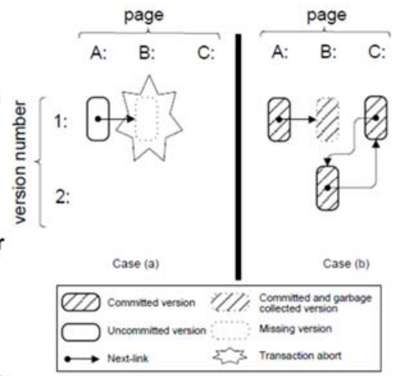
Peek into Architecture:

- Addition of new functionality to firmware
  - Commit, Garbage Collection, Recovery Logic
- Needs about 25% more memory for transaction tracking
- Needs different interface than native Disk interface
  - WriteAtomic, Abort



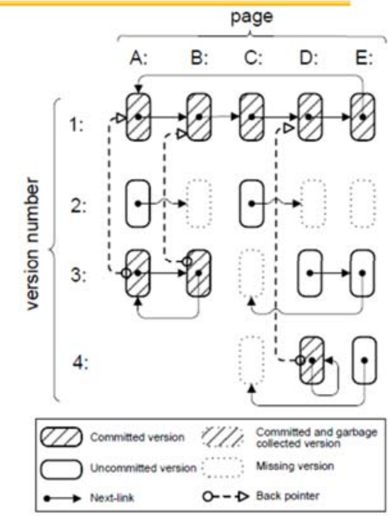
Simple Cyclic Commit (SCC)

- **Every flash page has:**
  - Page # (logical page)
  - Version # (monotonically increasing)
  - Pointer (called *next*) to another flash page (Page #,Version#)
  - Notation: P_j is j^{th} version of page P
- **Two key sets:**
  - Let *S* be set of existing records
  - Let *R* be set of records pointed at by other records (may not exist)
- **Cycle Property:**
  - For any intention record $r \in S$, *r* is committed ⇔ *r.next* is committed
  - If there is a complete cycle, then everyone in cycle is committed
- **SCC Invariant:**
  - If $P_j \in S$, any intention record $P_i \in S \cup R$ with i<j must be committed
  - Consequence: must erase failed commits before committing new versions of page



Back Pointer Cyclic Commit (BPCC)

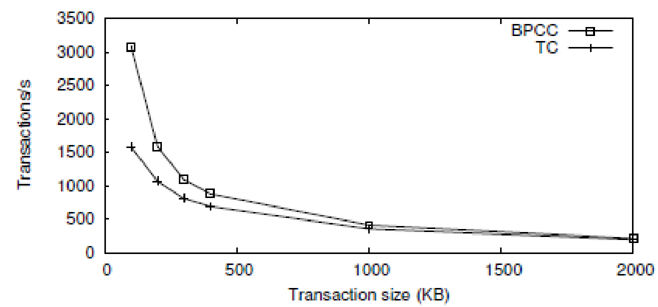
- **Introduce new piece of metadata: backpointer**
  - Points to most recent committed version of same page
  - Allows clear identification of failed commits by noticing intervening blocks which must be uncommitted
- **Complexity is all about garbage collection now**
- **Straddler**
  - For any record P_j: existence of P_k with P_k.*back* = P_i and i < j < k means that P_k straddles P_j
  - Means P_j is not committed!
- **BPCC Invariant:**
  - For a highest version intention record $P_h \in S$, Let $Q_l = P_h$.*next.* If there exists a $Q_k \in S$ with k > l and there exists no straddler for Q_l, then P_h is committed



Evaluation?

- **Model Checking of SCC and BPCC protocols**
  - Published elsewhere
- **Collect Traces from version of Ext3 (TxExt3) running on linux with applications**
  - This got them most of the way, but Ext3 doesn't really abort much
- **Synthetic Workload generator to generate a variety of transactions**
- **Flash Simulator**
  - SSD simulator from previous work described elsewhere
    - » Would have to look it up to know full accuracy
    - » Give them benefit of doubt
  - 32GB TxFlash device with 8 fully-connected 4GB flash packages
  - Parameters from Samsung data sheet

Savings from avoidance of commit



- **Log and data combined together**
- **By avoiding last commit record, have one less write**

General throughput results

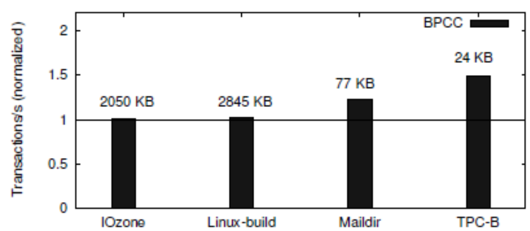


Figure 7: **Performance Improvement in Cyclic Commit.** *Transaction throughput in BPCC, normalized with respect to the throughput in TC. The throughput of IOzone, Linux-build, Maildir, and TPC-B in TC are 31.56, 37.96, 584.89, and 1075.27 transactions/s. The average transaction size is reported on top of each bar.*

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

Break

Facebook Reprise: How to Store Every Photo Forever?

- **82% of Facebook traffic goes to 8% of photos**
  - **Sequential writes, but random reads**
  - **Shingled Magnetic Recording (SMR) HDD with spin-down capability is most suitable and cost-effective technology for cold storage**
- **New Facebook datacenter in Prineville, OR**
  - **3 data halls, each with 744 Open Racks**
  - **1Open Vault storage unit holds 30 3.5" 4TB SMR SATA disks**
  - **1Open Rack holds 16 OV storage units (16 x 30 drives = 480 drives)**
  - **1 disk rack row has 24 Open Racks (24 x 480 drives = 11,520 drives)**
  - **1 data hall has 30 disk rack rows (30 x 11,520 drives = 345,600 drives)**
  - **Using 4TB SMR drives (4TB x 345,600 drives) = 1,382,400TB**
  - **3 data halls = 4.15 ExaBytes of raw capacity!!**

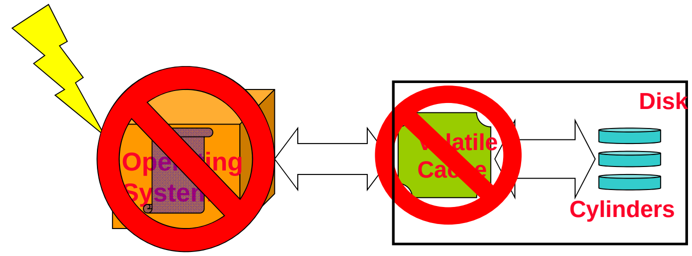
http://www.opencompute.org/wp/wp-content/uploads/2013/01/Open_Compute_Project_Cold_Storage_Specification_v0.

Rethink the Sync: Premise (Slides borrowed from Nightingale)

- Asynchronous I/O is a poor abstraction for:
  - Reliability
  - Ordering
  - Durability
  - Ease of programming
- Synchronous I/O is superior but 100x slower
  - Caller blocked until operation is complete
- New model for synchronous I/O: External Synchrony
  - Synchronous I/O can be fast!
  - Same guarantees as synchronous I/O
  - Only 8% slower than asynchronous I/O

When a sync() is really async

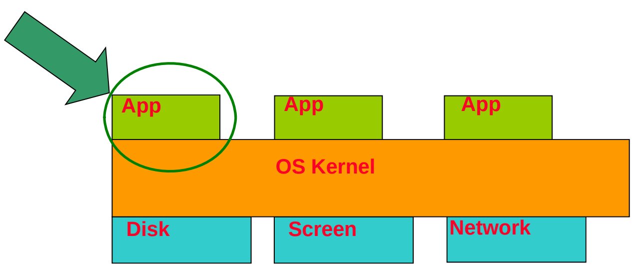
- On sync() data written only to volatile cache
  - 10x performance penalty and data NOT safe



- 100x slower than asynchronous I/O if disable cache

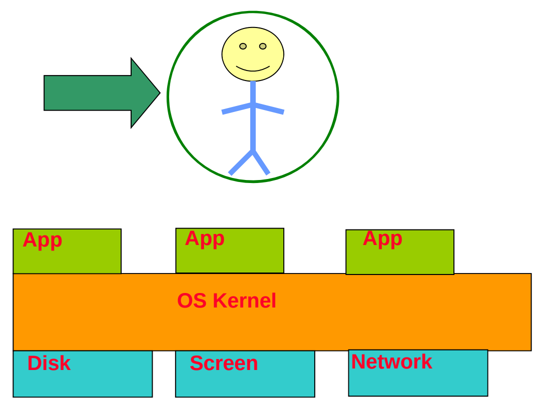
To whom are guarantees provided?

- Synchronous I/O definition:
  - Caller blocked until operation completes



- Guarantee provided to application

To whom are guarantees provided?

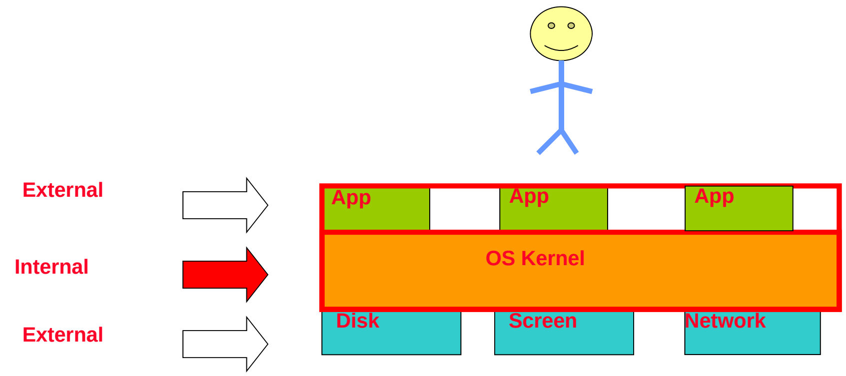


- Guarantee really provided to the user

Providing the user a guarantee

- User *observes* operation has completed
  - User may examine screen, network, disk…
- Guarantee provided by synchronous I/O
  - Data durable when operation observed to complete
- To observe output it must be externally visible
  - Visible on external device

Why do applications block?



- Since application external we block on syscall
- Application is internal: no need to block!

A new model of synchronous I/O

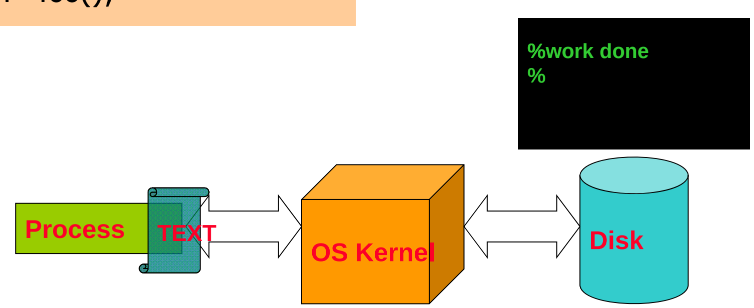
- Provide guarantee directly to user
  - Rather than via application
- Called externally synchronous I/O
  - Indistinguishable from traditional sync I/O
  - Approaches speed of asynchronous I/O

Example: Synchronous I/O

```
101   write(buf_1);
102   write(buf_2);
103   print("work done");
104   foo();
```

Application blocks

Application blocks



Observing synchronous I/O

```
101   write(buf_1);
102   write(buf_2);
103   print("work done");
104   foo();
```

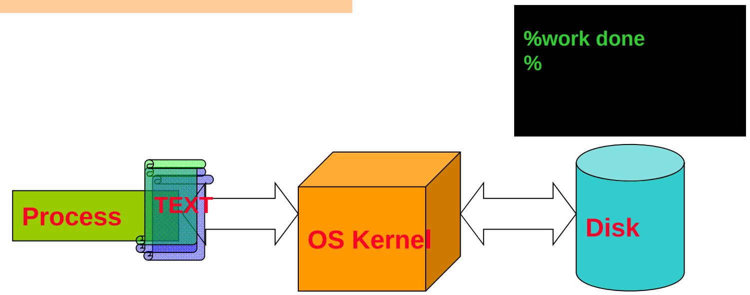
Depends on 1st write

Depends on 1st & 2nd write

- Sync I/O externalizes output based on causal ordering
  - Enforces causal ordering by blocking an application
- Ext sync: Same causal ordering without blocking applications

Example: External synchrony

```
101   write(buf_1);
102   write(buf_2);
103   print("work done");
104   foo();
```



Tracking causal dependencies

- Applications may communicate via IPC
  - Socket, pipe, fifo etc.
- Need to propagate dependencies through IPC
- Authors build upon Speculator [SOSP '05]
  - Track and propagate causal dependencies
  - Buffer output to screen and network
  - Targeted at improving performance when network is involved
    - » (Such as for a Network File System)
  - Return immediately with speculative result
    - » Checkpoint processes, restore checkpoint if real result doesn't match speculated result
- Pieces of Speculator useful here:
  - Tracking of dependencies to make sure that we maintain property of External Synchrony
- I've put up the SOSP 2005 paper as an optional reading

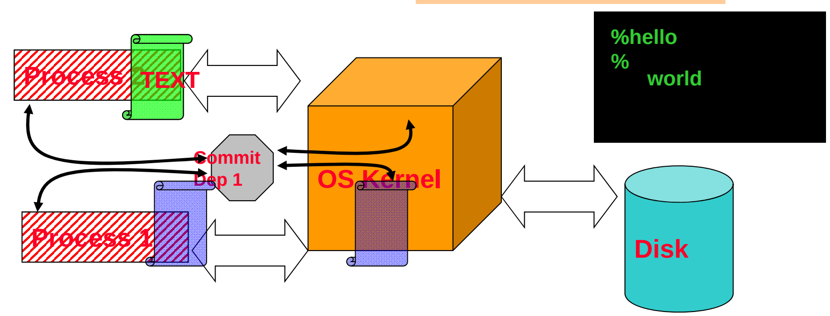
Tracking causal dependencies

Process 1

```
101   write(file1);
102   do_something();
```

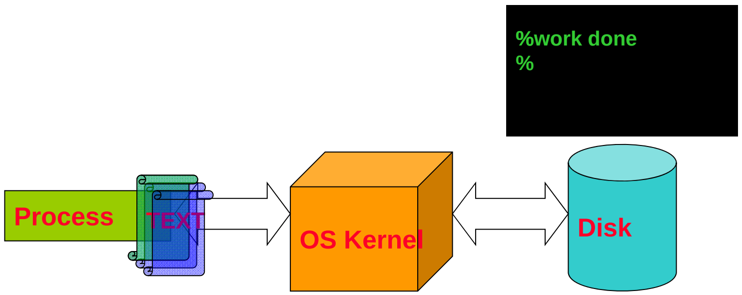
Process 2

```
101   print ("hello");
102   read(file1);
103   print("world");
```



Output triggered commits

- **Maximize throughput until output buffered**
- **When output buffered, trigger commit**
  - **Minimize latency only when important**



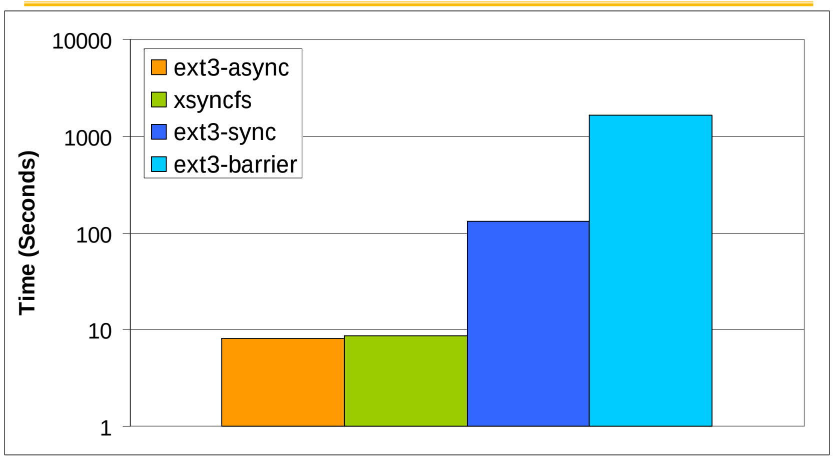
Evaluation

- **Implemented ext sync file system <u>Xsyncfs</u>**
  - **Based on the ext3 file system**
  - **Use journaling to preserve order of writes**
  - **Use write barriers to flush volatile cache**
- **Compare Xsyncfs to 3 other file systems**
  - **Default asynchronous ext3**
  - **Default synchronous ext3**
  - **Synchronous ext3 with write barriers**

When is data safe?

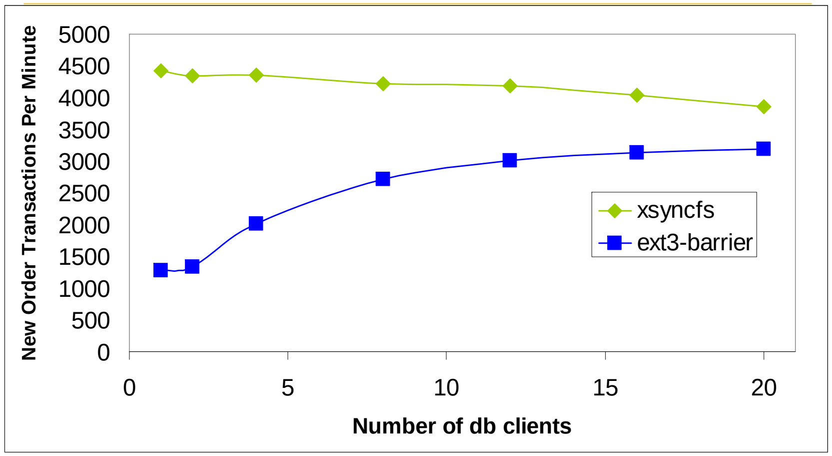
File System Configuration	Data durable on write()	Data durable on fsync()
Asynchronous	No	Not on power failure
Synchronous	Not on power failure	Not on power failure
Synchronous w/ write barriers	Yes	Yes
External synchrony	Yes	Yes

Postmark benchmark



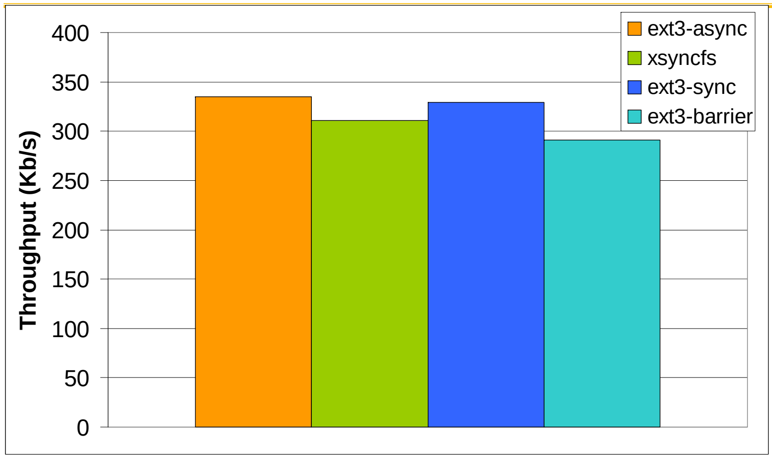
- **Xsyncfs within 7% of ext3 mounted asynchronously**

The MySQL benchmark



- **Xsyncfs can group commit from a single client**

Specweb99 throughput



- **Xsyncfs within 8% of ext3 mounted asynchronously**

Specweb99 latency

Request size	ext3-async	xsyncfs
0-1 KB	0.064 seconds	0.097 seconds
1-10 KB	0.150 second	0.180 seconds
10-100 KB	1.084 seconds	1.094 seconds
100-1000 KB	10.253 seconds	10.072 seconds

- **Xsyncfs adds no more than 33 ms of delay**

Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?