Memory-Centric Data Storage for Mobile Systems

Jinglei Ren, Mike Liang, Yongwei Wu,
Thomas Moscibroda

清華大學
Tsinghua University

Microsoft Research

Two things you may dislike most about your smartphone…

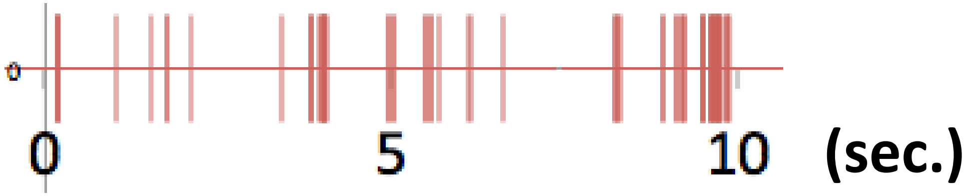
Battery drain

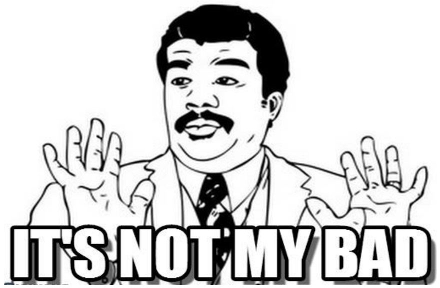
Low responsiveness

But do you know…

What
Is an app doing
Behind you?!

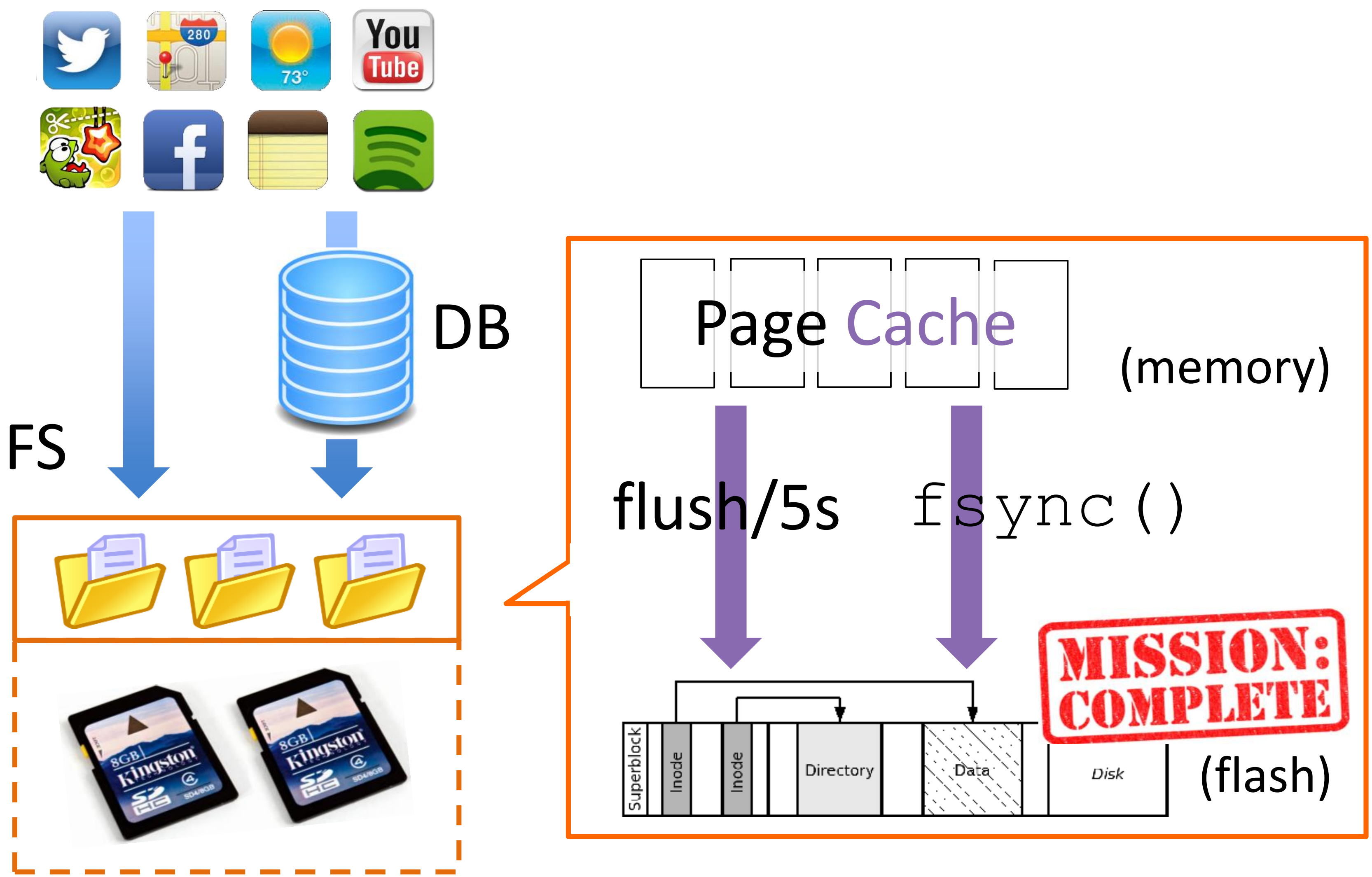
Twitter's fsync() system calls





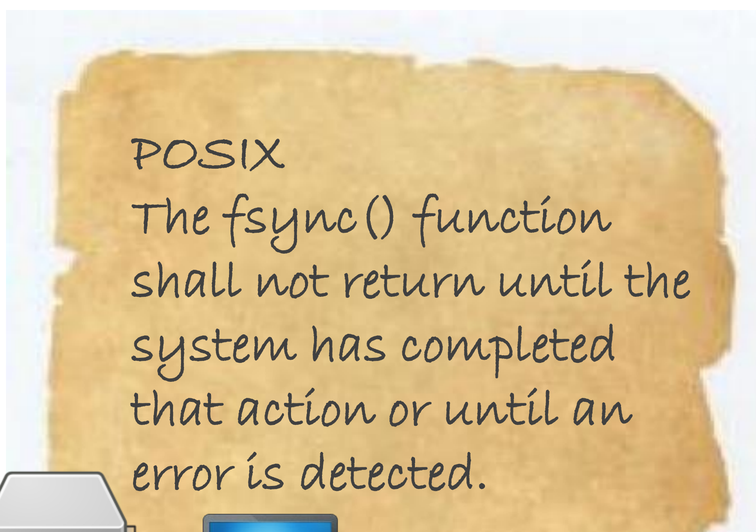
Storage impairs both energy efficiency and responsiveness!

Traditional Design

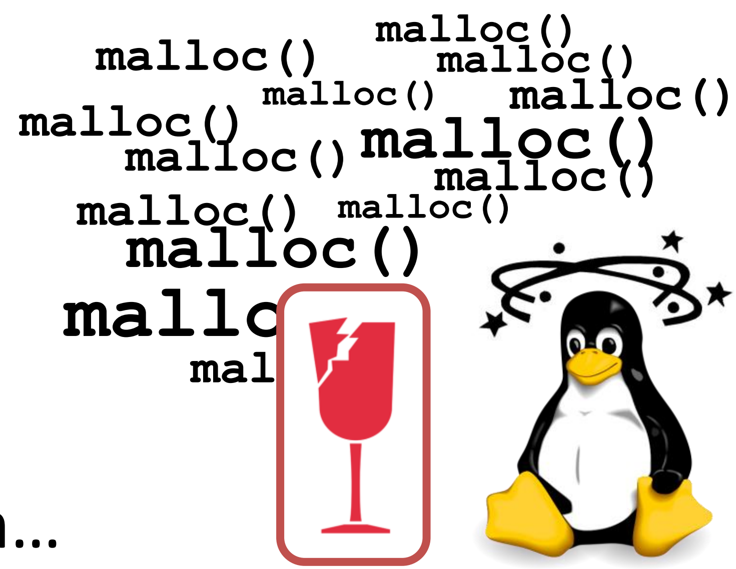


Traditional Design

Programmers’ dilemma

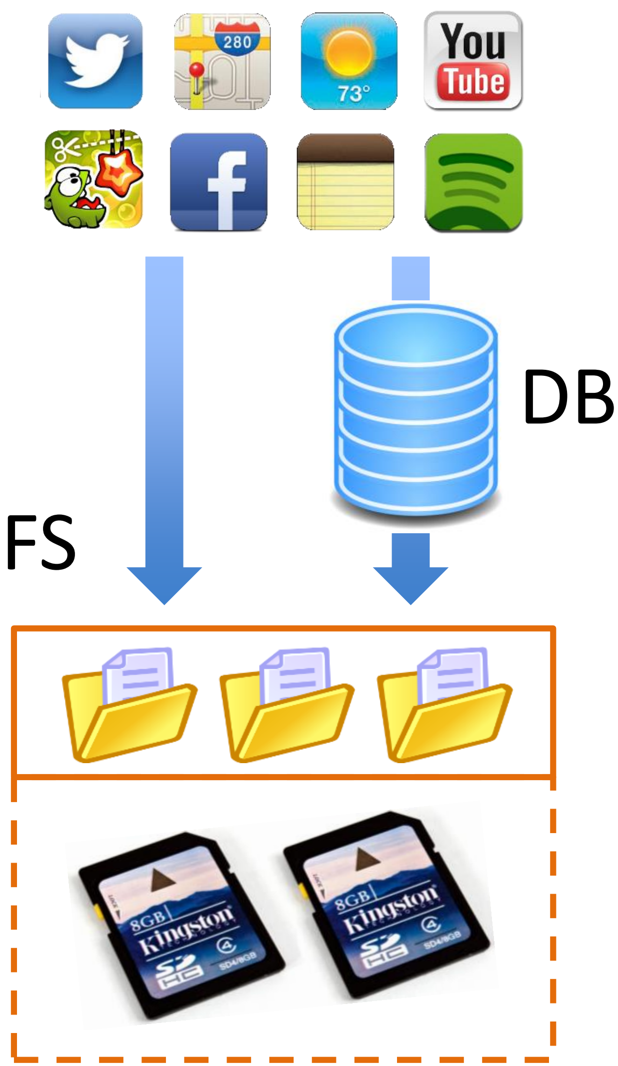


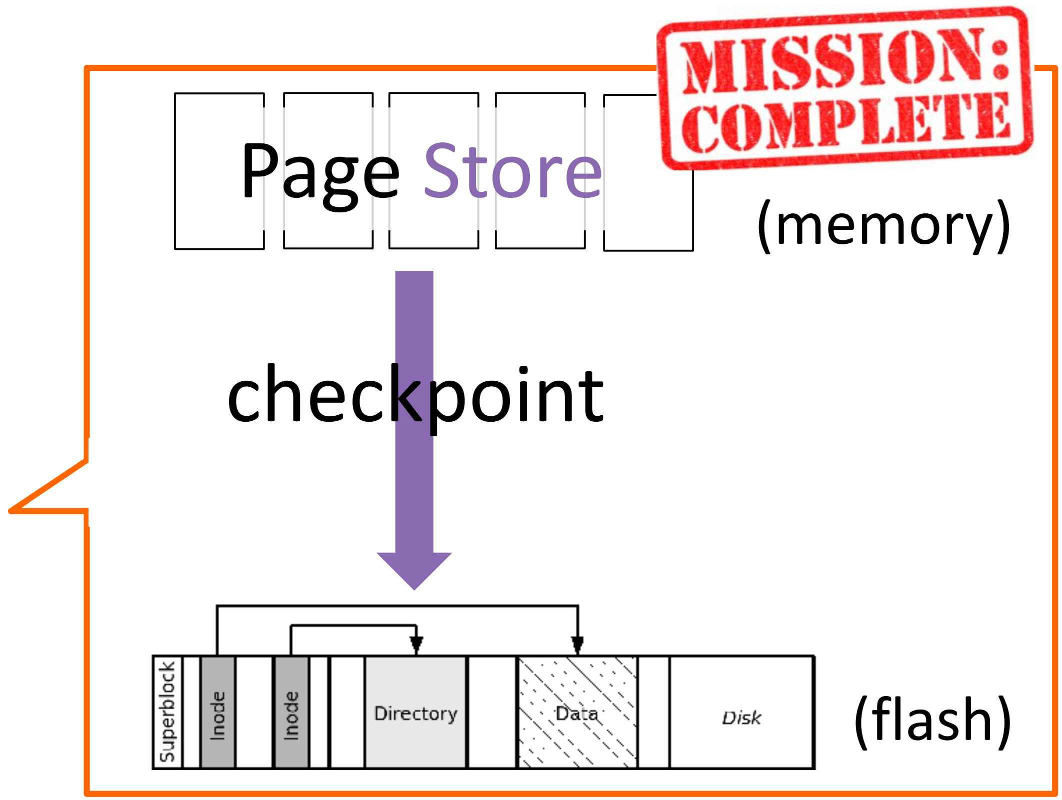
Old-fashioned design…



Solution Overview

Flash storage vs. DRAM residence: Can we find a sweet spot between the two?





Insight I

Storing app data on smartphone memory is not as risky as it sounds.

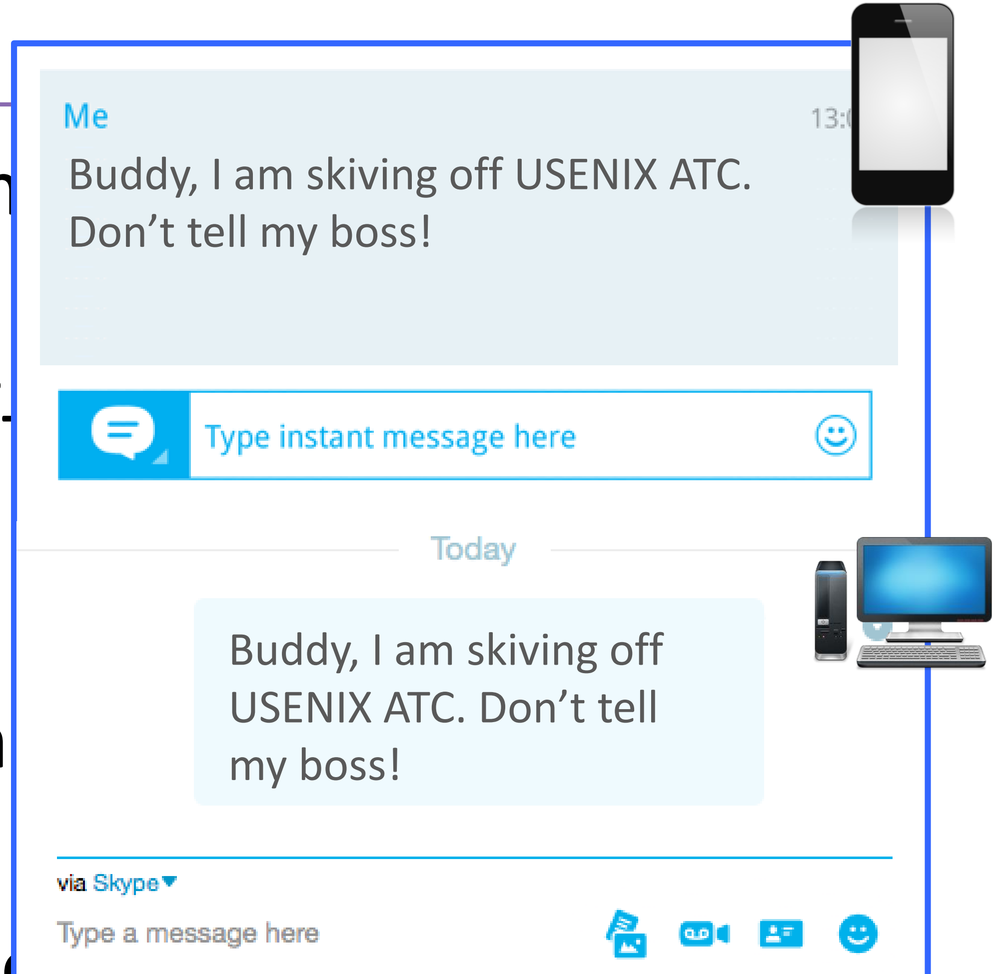
- A smartphone is self-contained,i.e., battery-backed.
- System-wise crash is rare. Our survey: only 6% users experienced more than once per month.
- Our case studies: 54 out of top 62 free apps in Google Play are vulnerable to local data loss.

About Pinterest

Privacy Policy

What information do we collect?

...This can include your name, profile photo, Pins, comments, likes, email address..., and any other information you provide us.



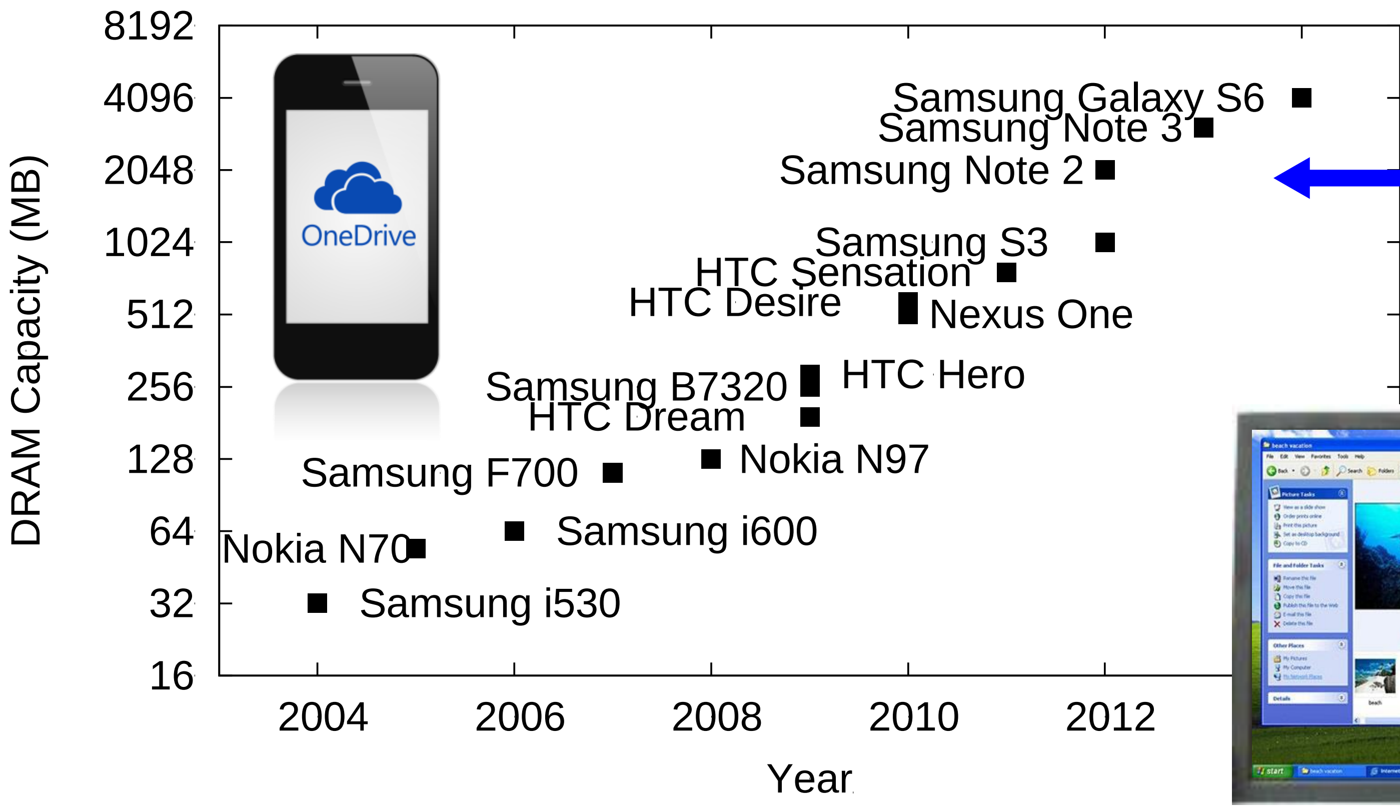
System Design: Mechanism

Versioned Cache Transaction (VCT)

- Introducing transactions to OS page cache
- Basic life cycle:
  - Open a VCT for certain files
  - Perform Copy-on-Write for dirty pages
  - Coalesce writes on these new versions of pages
  - Close a VCT according to our policy
- VCTs of different apps are independent, for optimization purpose.

Insight II

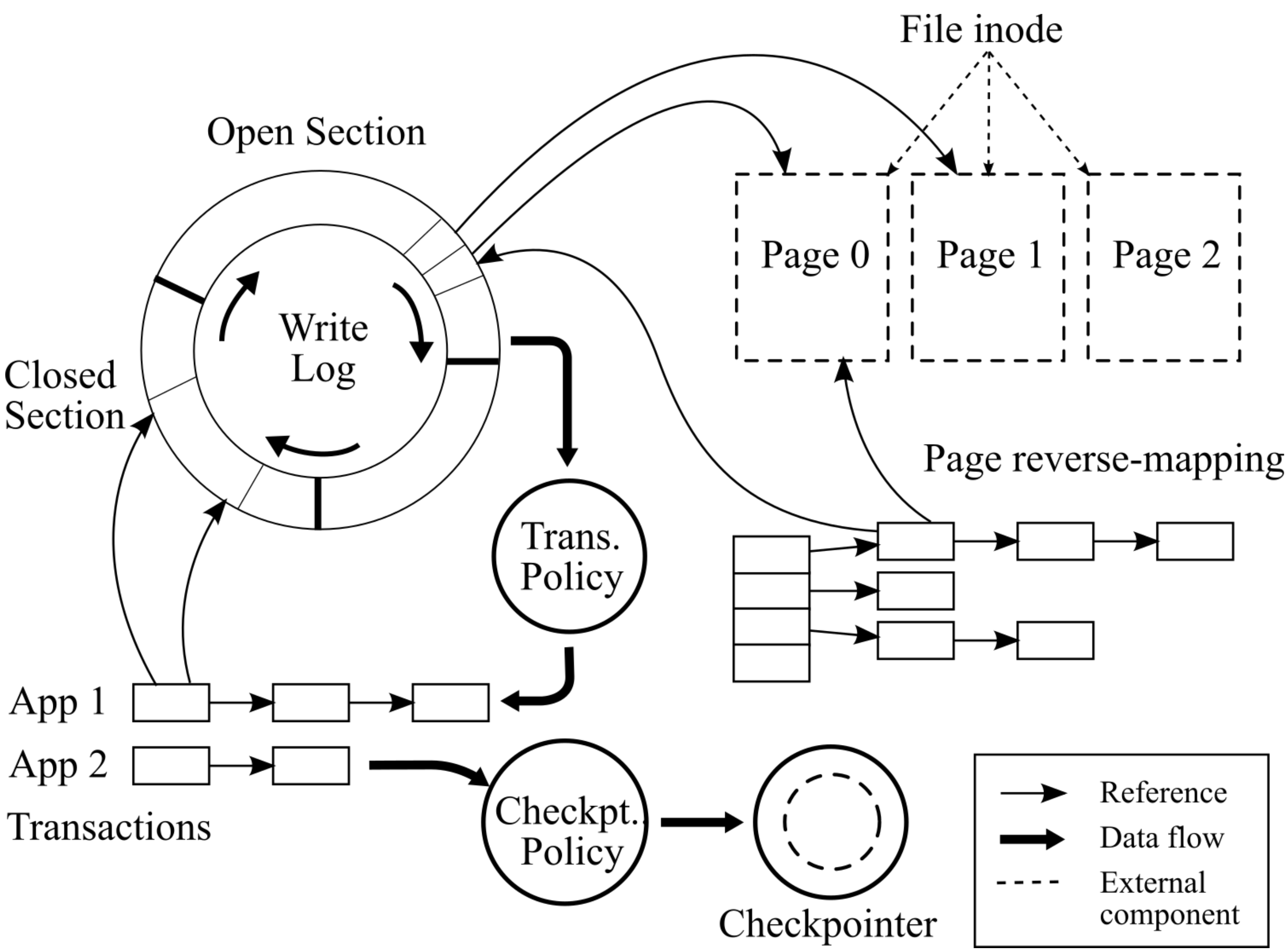
Memory capacity on smartphones is ample enough for app data storage.



Microsoft Windows xp

System Design: Mechanism

MobiFS components

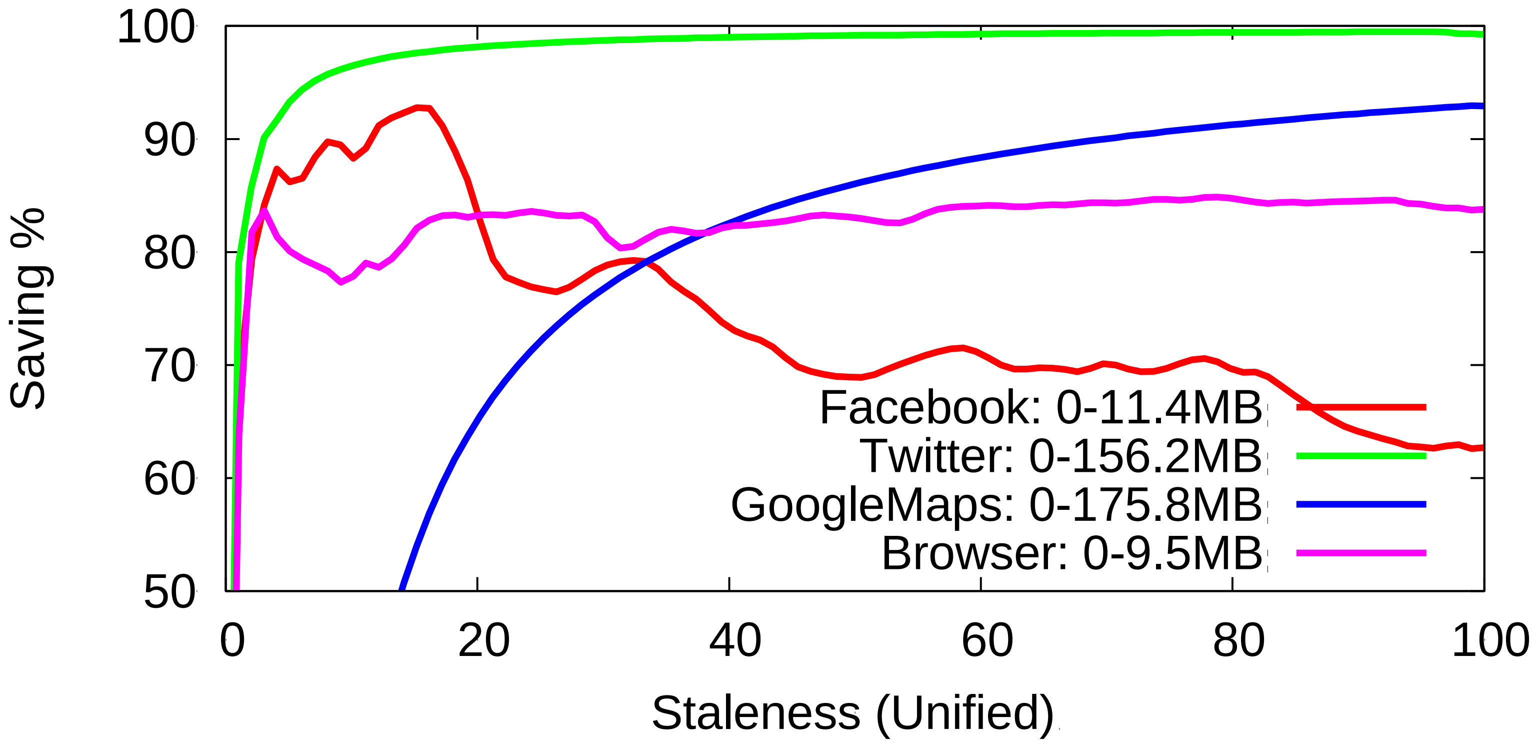


Insight III

Reducing the amount of data flushed to flash is a key to save app energy.

- Our measurement: the overall read energy is only 6.3% of write energy
- The amount of data to flush, rather than the number of batches, is the dominant factor. Our measurement: writing 40 MB data in batches ranging from 4 to 40 MB results in a net energy consumption difference within 1.5%.

Insight V



App I/O patterns suggest adaptive policies to balance the staleness-energy tradeoff, which can be achieved in a quantitative way.

System Design: Policy

Tradeoff Point Location

- New metric for energy efficiency: the e curve
  $e =$ coalesced data size / staleness
- Principle: reduce data staleness unless the otherwise increases energy efficiency.
- Peak detection algorithm:
  - Detection window
  - Incremental linear regression
  - Threshold for gradient (not necessarily 0)

System Design: Policy

- Tradeoffs between three objectives: data staleness, energy efficiency and app responsiveness.
- The tradeoff point location algorithm only closes a transaction, making it ready to be checkpointed.

Responsiveness-oriented policy: when to ckpt.

Insight IV

Relaxing the timing of flushes is a key to app responsiveness.

- Prior work has shown the implication of fsync() [Jeong et al. ATC'13, Lee et al. EMSOFT'12] and background flushing [Kim et al. FAST'12, Nguyen et al. UbiComp'14].
- What is the right timing for flushing?
  Our measurement: when the device is idle.
  Standby is *not* good timing – leading to 129% extra energy consumption

System Design: Policy

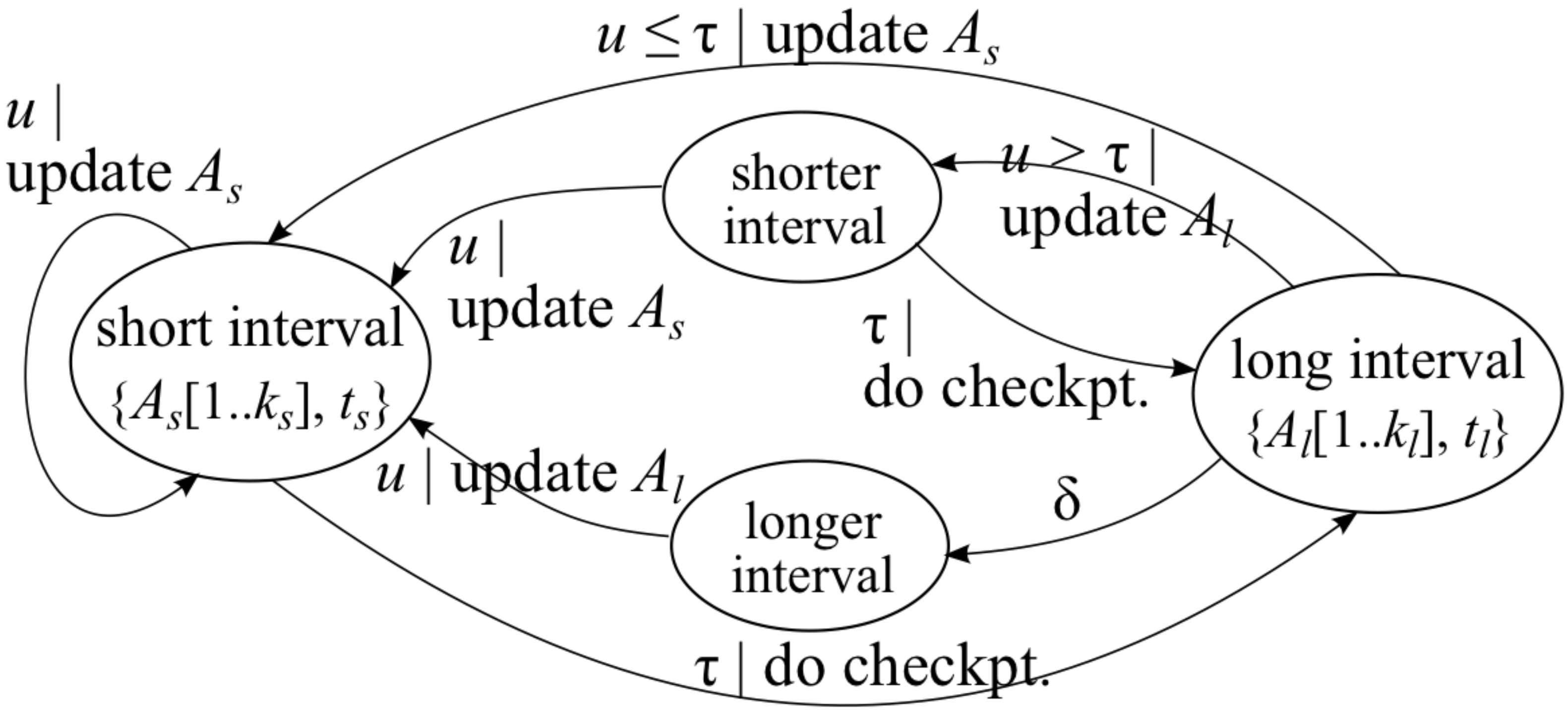
Interval Prediction

- Rationale: predict according to history
- Last min policy
  - Pessimistic in prediction, with least conflicts
  - Limiting flush data size
- Last average policy
  - Incurring more conflicts
  - Enabling larger flush data size

System Design: Policy

Interval Prediction

- To learn two modes in user interaction



event u - an user operation
event τ - when $m \times t_s$ passes; event δ - when t_l passes

Implementation and Evaluation

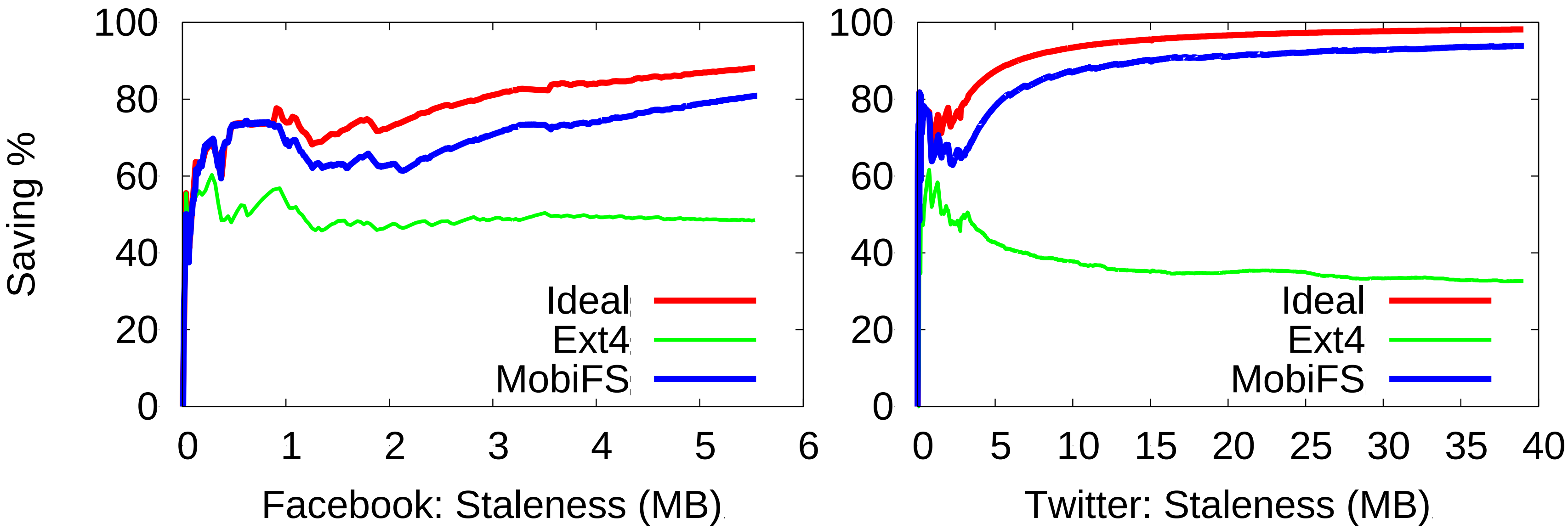
A working prototype

- Android 4.1 (Linux 3.0.31)
- Integrated with either Ext4 (journaling data) or Btrfs (COW)

Experiments

- Traces from real users
- Benchmarks + real apps (monkeyrunner)
- Use real devices: Samsung Galaxy Premier I9260 (dual-core 1.5 GHz CPU, 1 GB RAM);

Evaluation: Energy

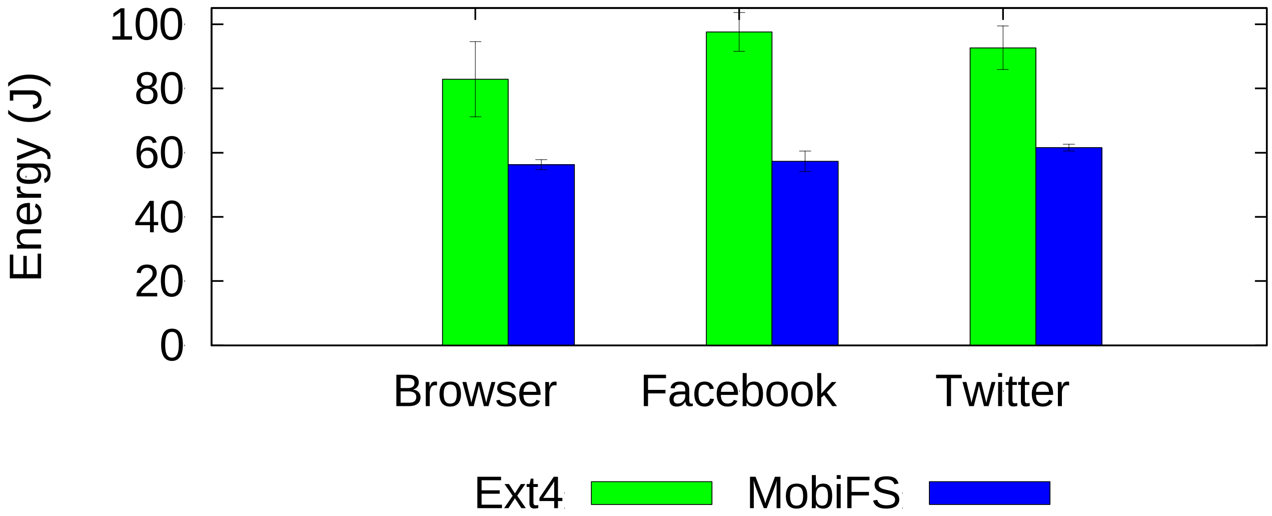


With ten most popular apps (by geo. mean):

- MobiFS reduces the amount of flush data by 53.0% compared to Ext4.

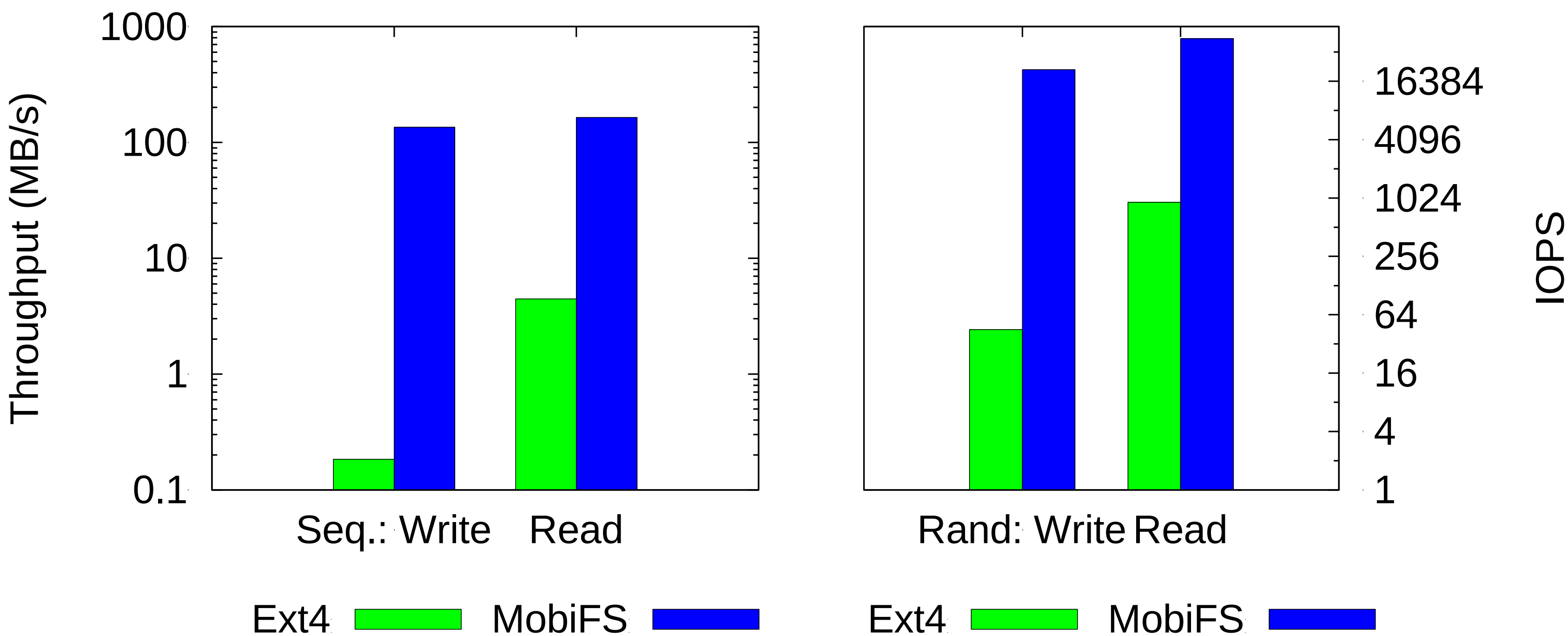
Evaluation: Energy

- Three representatives of real apps: Browser (low freq. of fsync), Facebook (middle freq. of fsync), Twitter (high freq. of fsync).



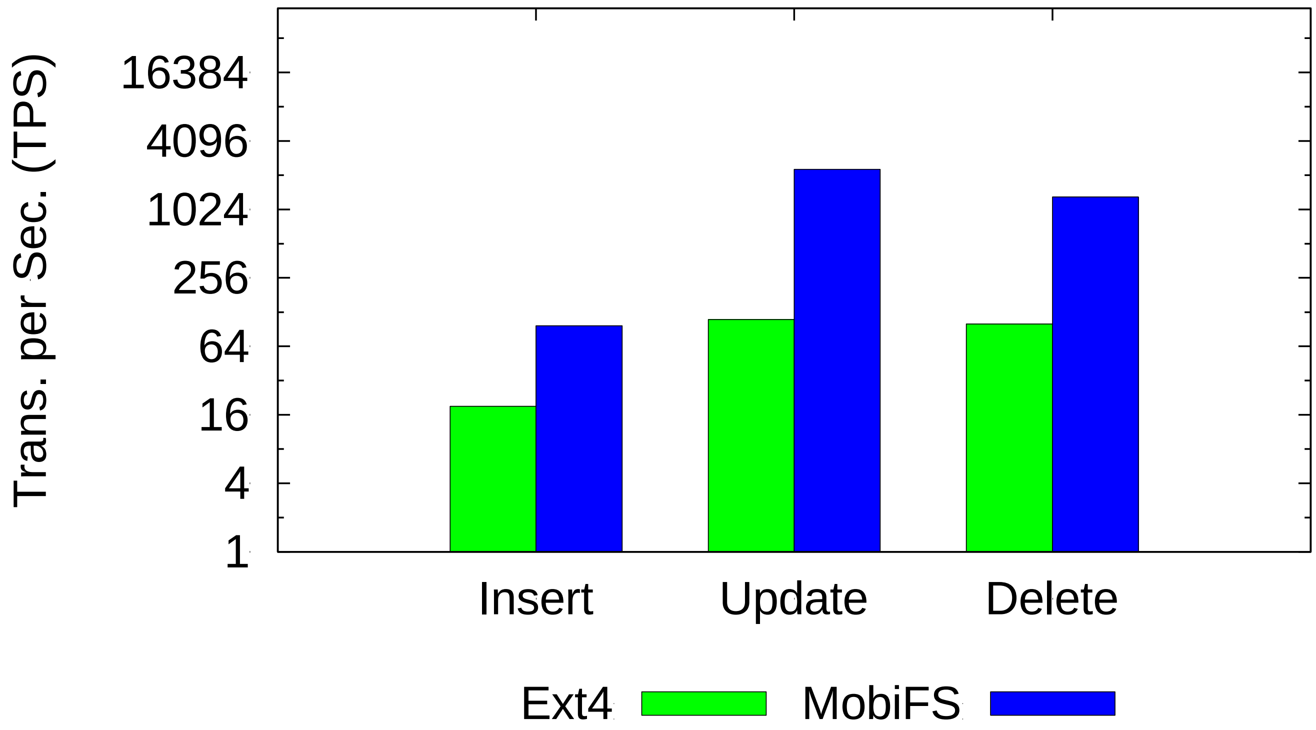
- On average, device energy consumption is reduced by 35.8% compared to Ext4.

Evaluation: Responsiveness



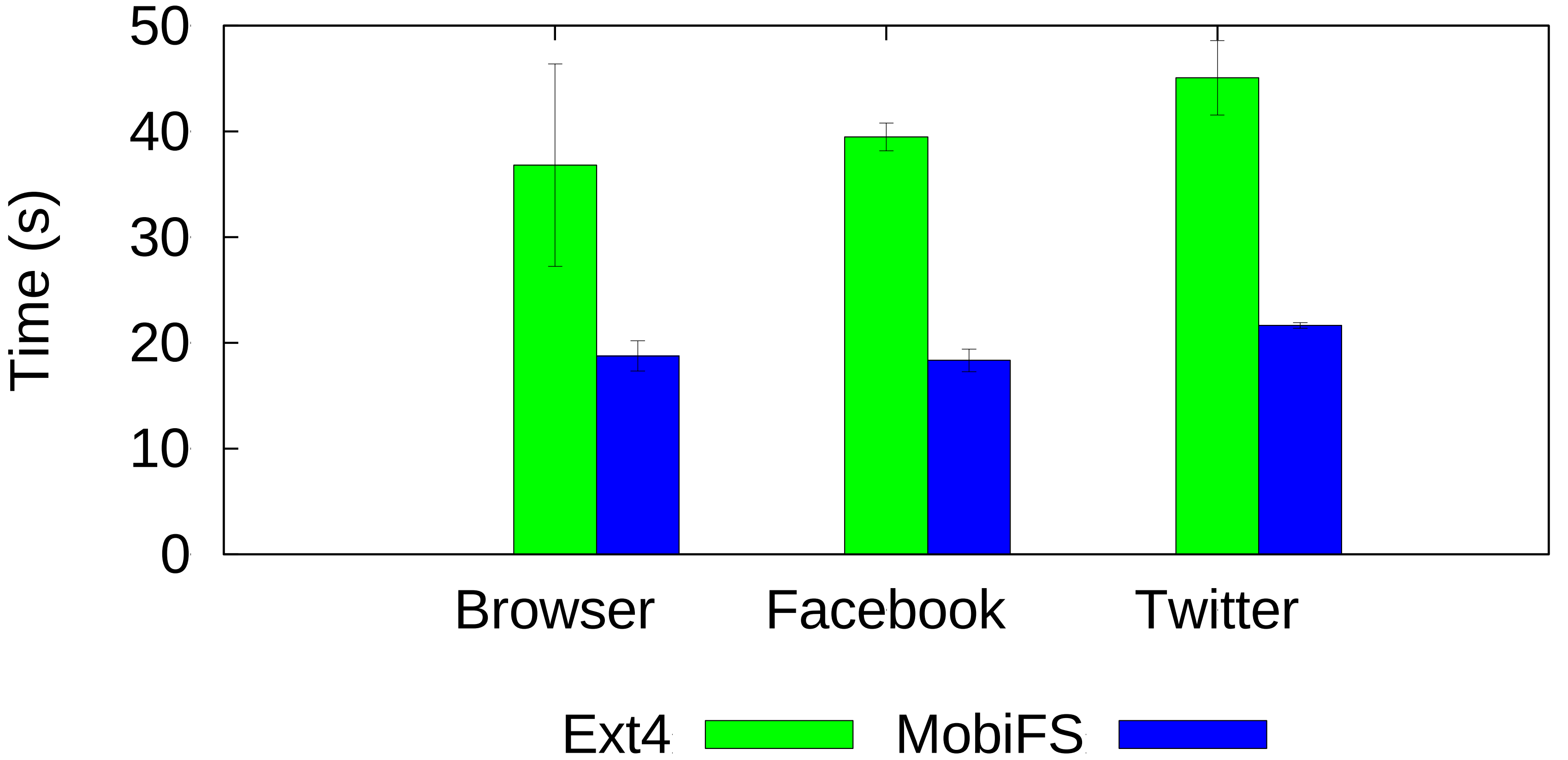
- On average, 18.8× filesystem I/O throughput.

Evaluation: Responsiveness



- On average, 11.2× database transaction throughput.

Evaluation: Responsiveness



- On average, user operation delay is reduced by 51.6%.

Related Work

- Decouple of durability and consistency

xsyncfs [OSDI'06], OptFS [SOSP'13], Blizzard [NSDI'14], TxCache [OSDI'10], etc.: different domains; static durability guarantee (e.g., up to x seconds of data loss).

MobiFS: transactions in OS page cache; adaptive tradeoff for different mobile apps/users.

- Energy optimizations

SmartStorage [UbiComp'13]: read/write ratio; 6% ~ 9% slowdown for energy saving

Coop-I/O [OSDI'02]: deferrable requests

MobiFS: changed design rationale; best performance

Conclusion

- We propose a memory-centric storage, based on our new insights in the mobile system design.
- We trade off data durability for energy efficiency and app responsiveness, in a quantitative manner.
- We introduce transactions to the OS page cache and implement MobiFS, to support the tradeoff transparently.
- We achieve: (1) over one order of magnitude improvement in IO performance; (2) over 1/2 and 1/3 reduction in energy consumption and operation delay, respectively.

Thank you!

jinglei@ren.systems