Intrinsyc Software

Linux on eMMC

Optimizing for Performance

Ken Tough
Principal Engineer
ktough@intrinsyc.com

What is eMMC?

- Solid state storage device on MMC bus
- Chip on PCB
- NAND flash based

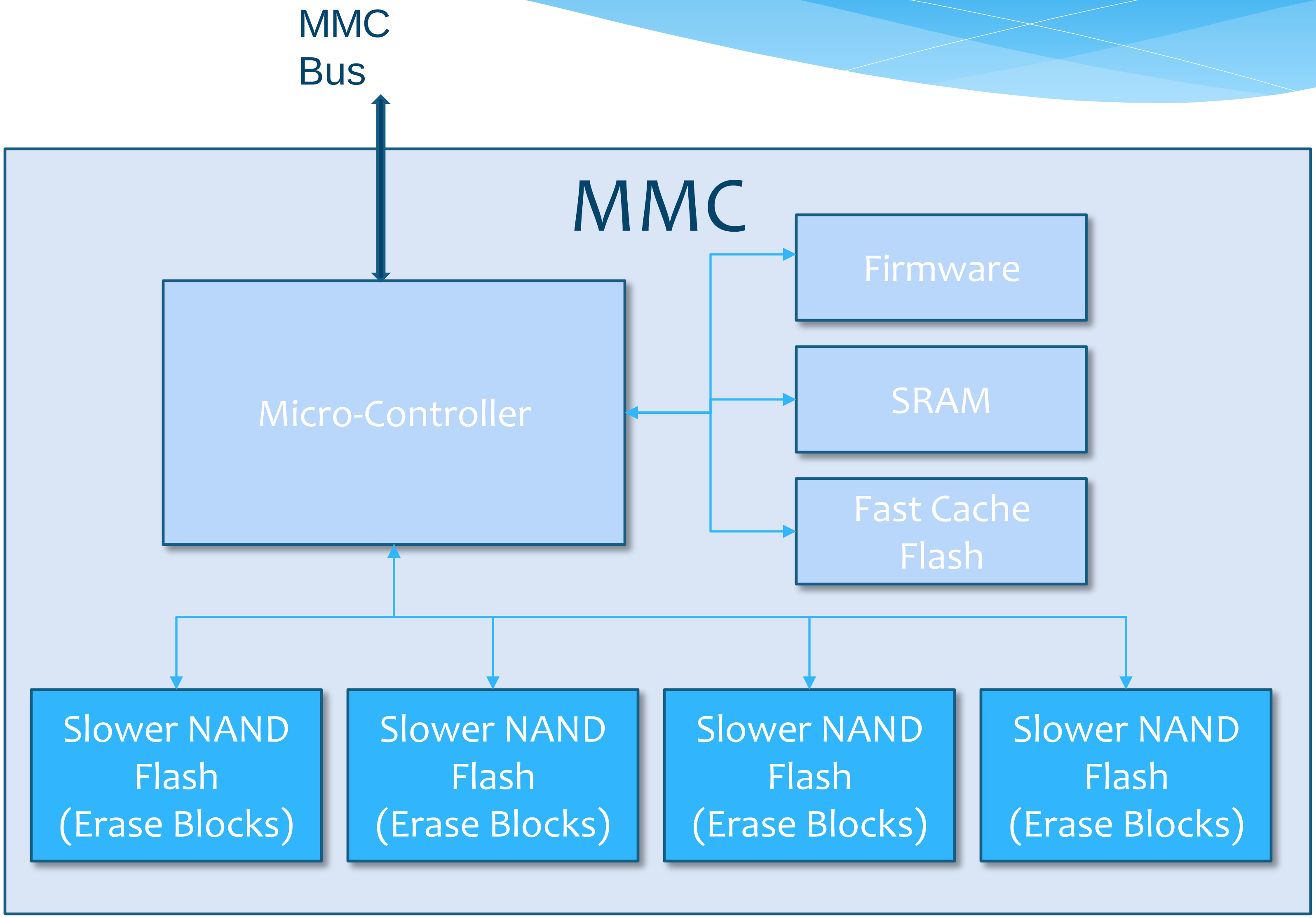
Why eMMC matters

- Popular on embedded devices
- Cheap
- Flexible

eMMC characteristics

- Fast read access
- Fast read seek times
- Acceptable sequential write performance
- Poor random write performance

Inside


MMC
Bus
MMC
Micro-Controller
Firmware
SRAM
Fast Cache Flash
Slower NAND Flash (Erase Blocks)
Slower NAND Flash (Erase Blocks)
Slower NAND Flash (Erase Blocks)
Slower NAND Flash (Erase Blocks)

Inside the eMMC

- NAND flash arranged in pages
- Controller with temporary storage
- Wear levelling
- Free space management

Discard (TRIM)

- eMMC TRIM command
- Tells controller what is free
- TRIM blocks on format

eMMC scenarios

- Tablets, smart phones with lots of DRAM
- Netbooks with lots of DRAM
- Multimedia players, USB memory sticks

eMMC spec performance

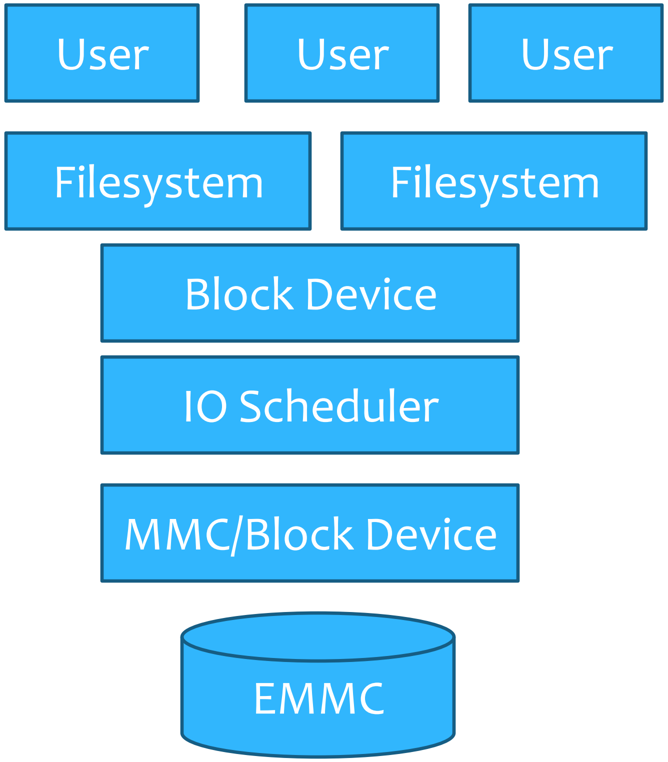
- Typically emphasizes sequential write performance
- Random accesses hit eMMCs internal pipelines
- Frequently limited by eMMC’s Random IOPs limit
- Minimum OP time regardless of OP size
- Not often data BW limited
- ~200 IOPs (e.g. 4kB per OP)
- Analyze application’s eMMC read/writes patterns

Cache is King

- Alleviates write performance issues
- Improves read times even further
- Reduces NAND wear

Areas of Focus

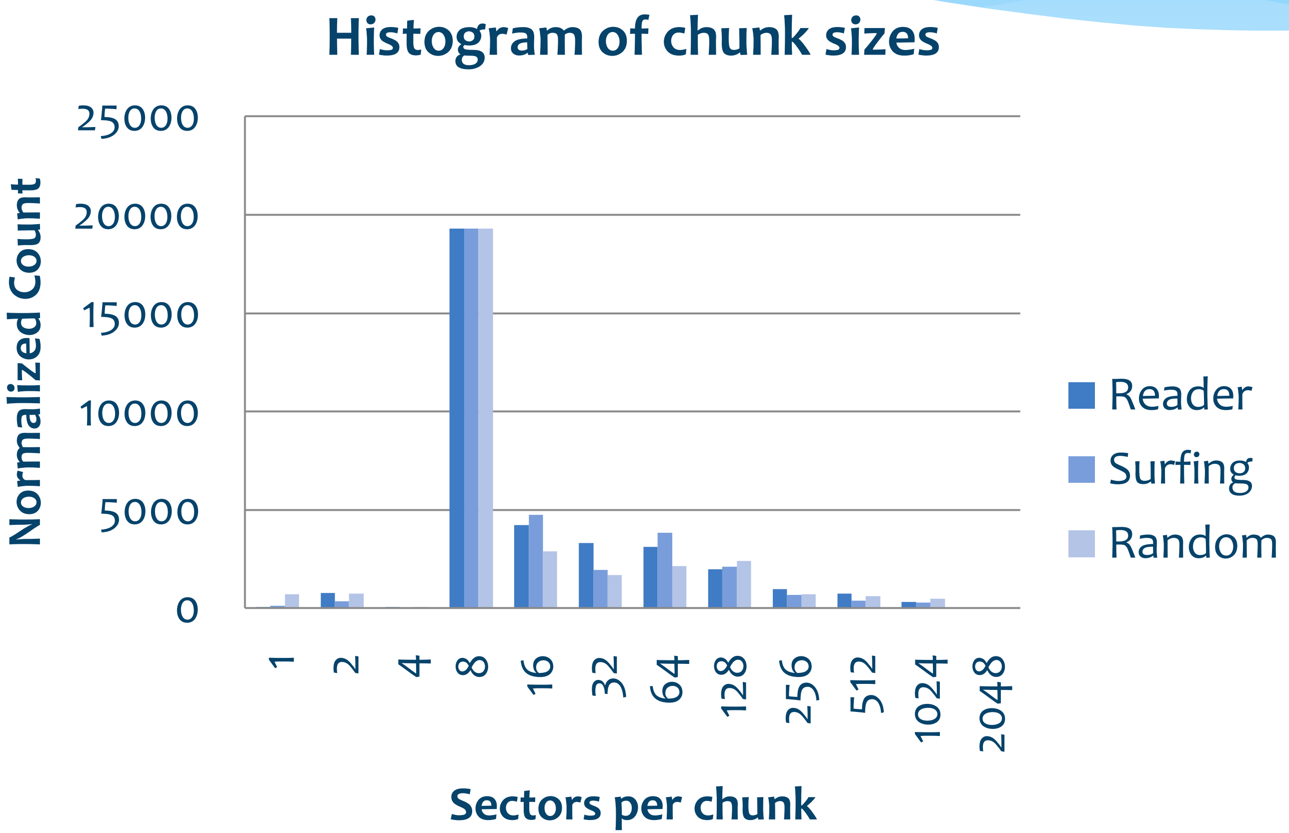
- User space
- Filesystem type
- Filesystem layout
- IO Scheduler
- Block IO & Cache
- MMC bus driver



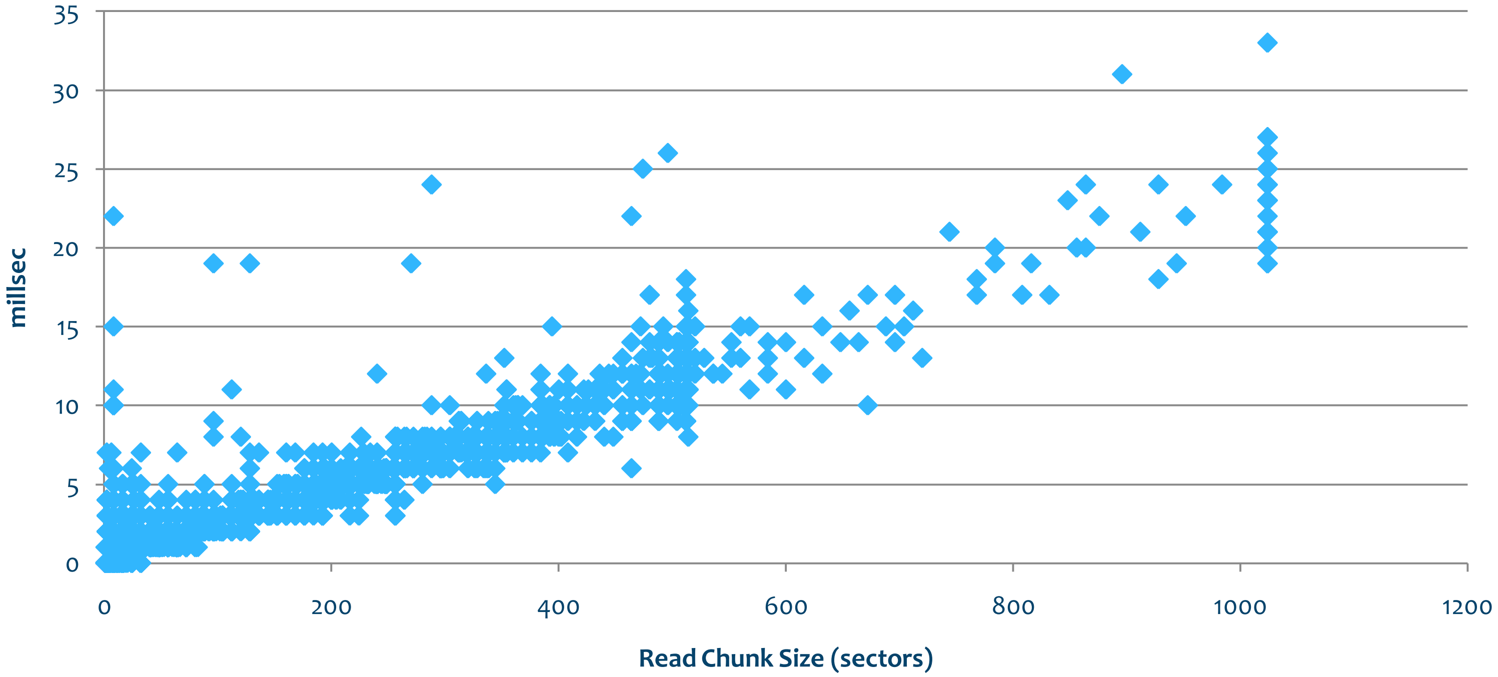
MMC driver

- Maximum bandwidth enabled (8-bit, 50MHz)
- Enable DMA if option
- Power management
- Trim / vendor command support
- Benchmarking Log

Analysis at MMC/Block Level

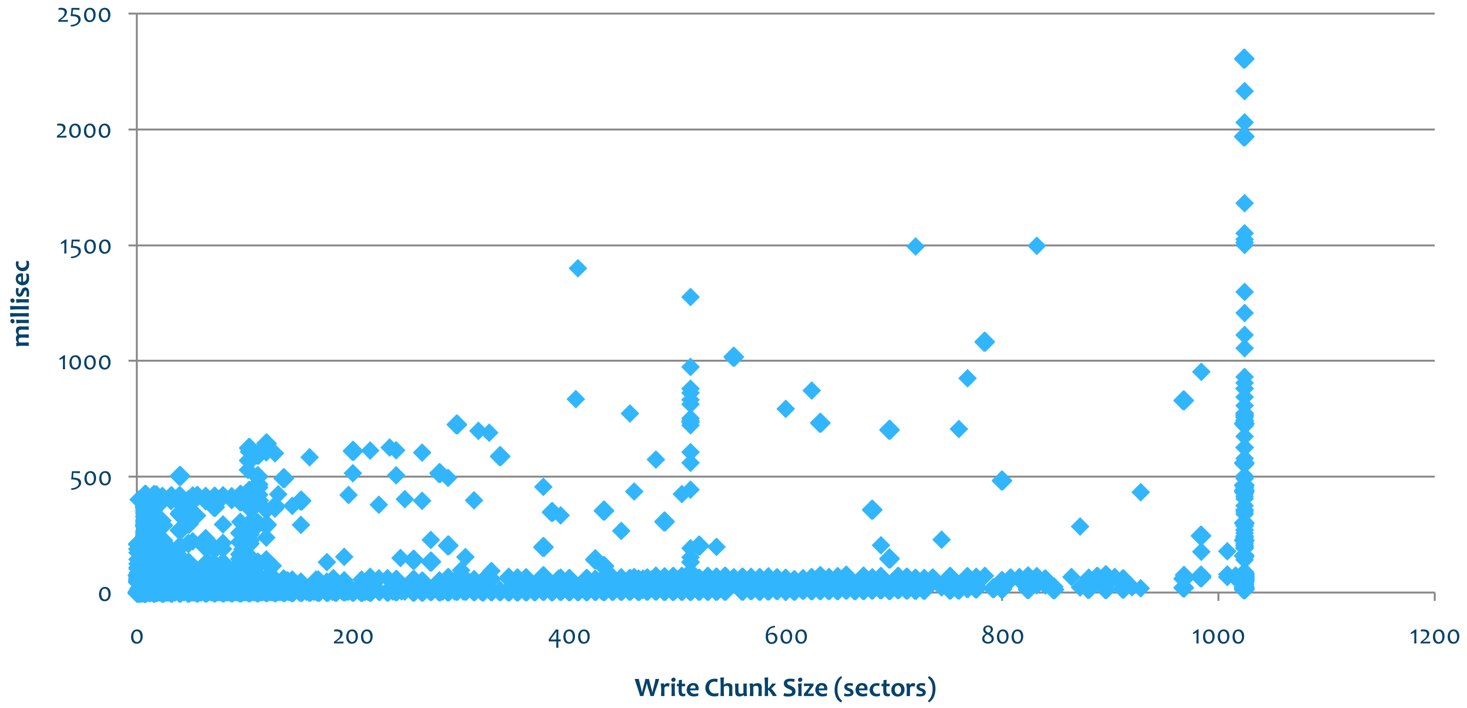


eMMC Read Times



CONFIDENTIAL INFORMATION

eMMC Write Times



Vendor Performance

- Wide variation in read/write times
- Big dependency on internal eMMC firmware
- Power Class support
- Geometry / technology
- Trim support

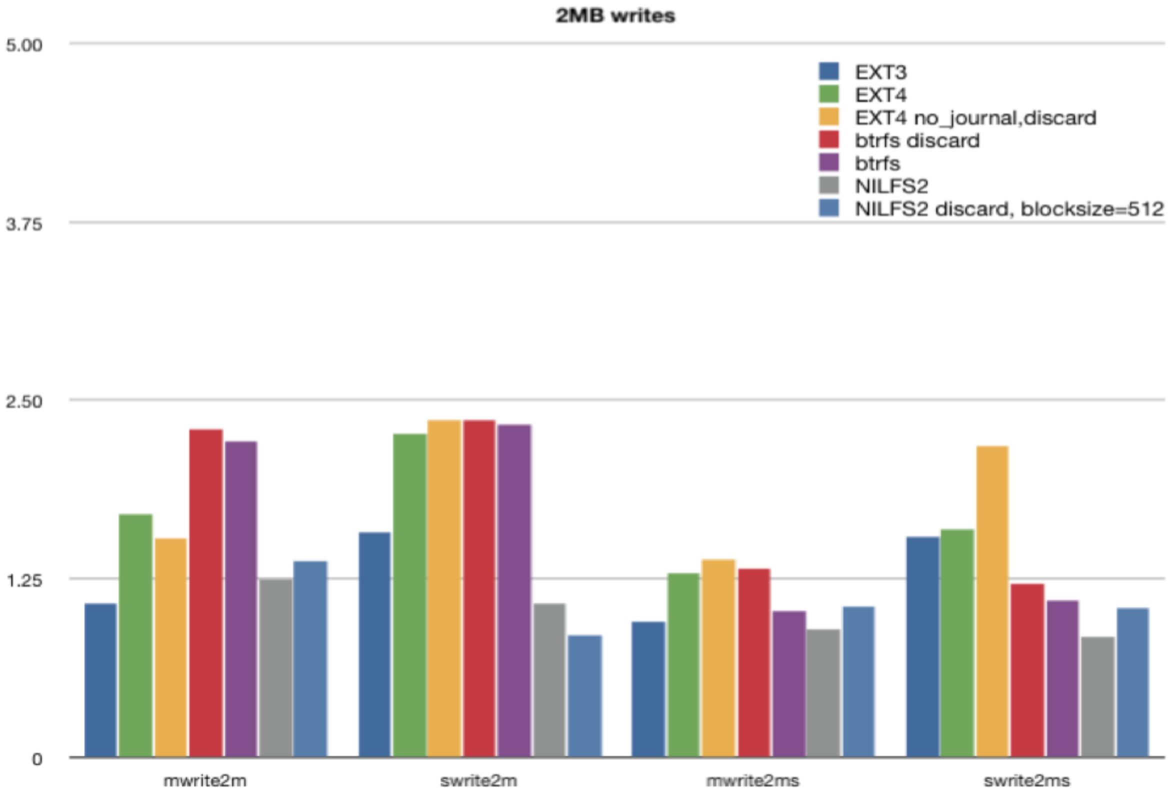
MMC v4 High Priority Interrupt

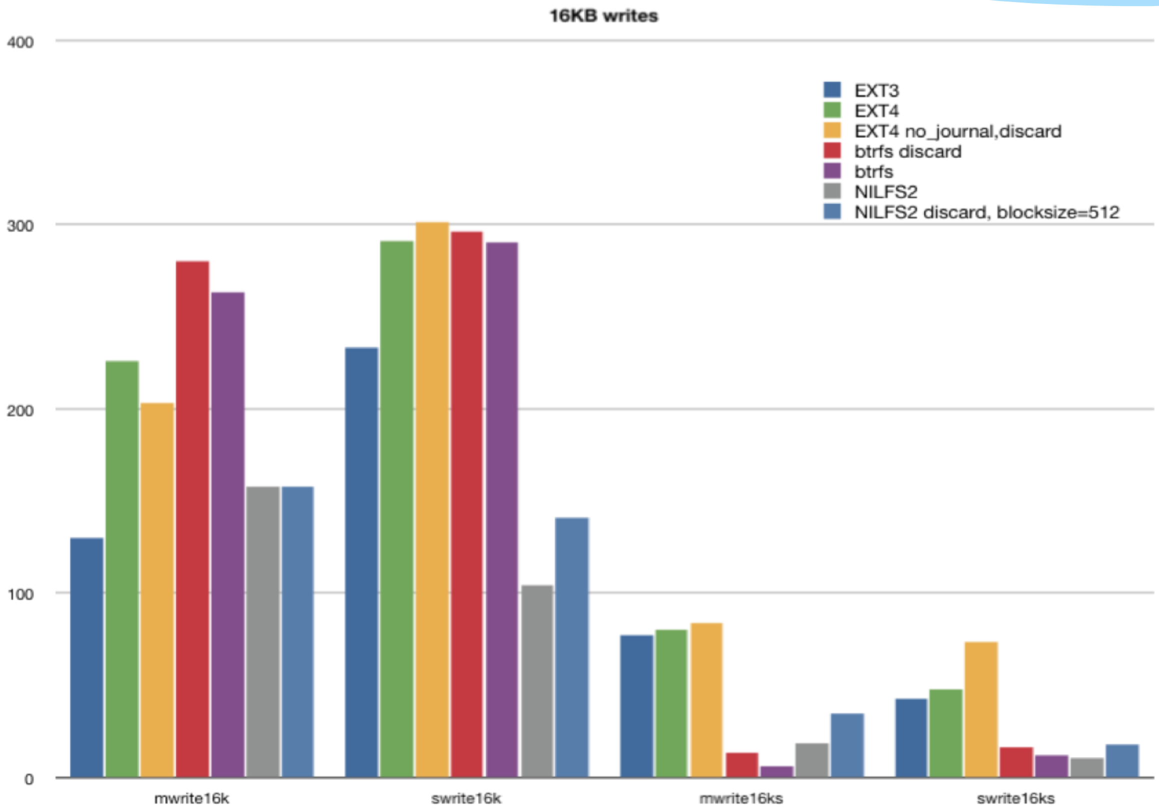
- Allows reads to bypass long writes
- Useful in very specific applications
- Small RAM
- Page/Block cache and IO Scheduler
- Internal eMMC Pipelines blocked anyway
- Multimedia apps and “long” buffering

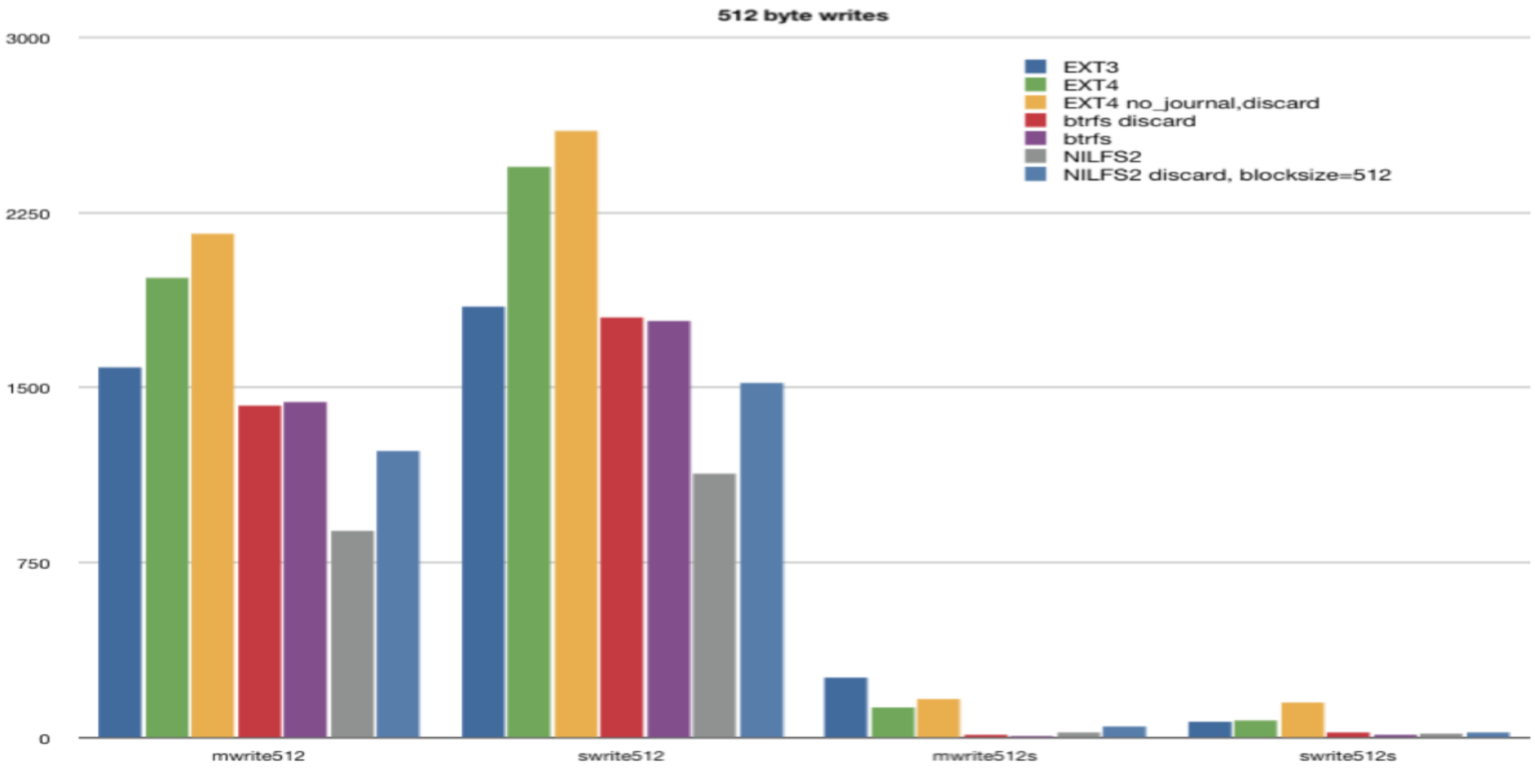
Filesystems

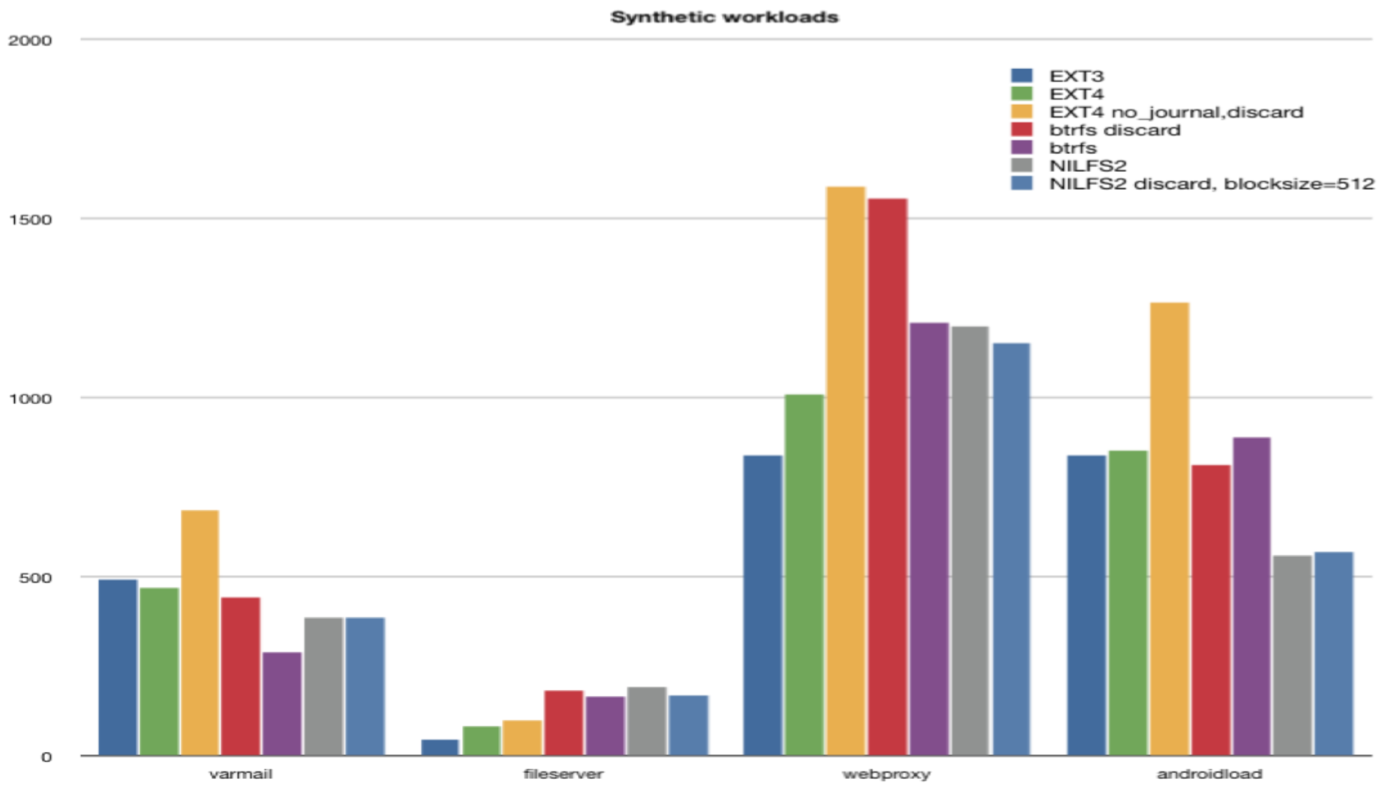
- Focus on write performance
- Tests run using fsbench (3.0 kernel, OMAP3 aka Nook Color)
- Various low-level and high-level scenarios modelled
- EXT4, BTRFS, NILFS2 tested

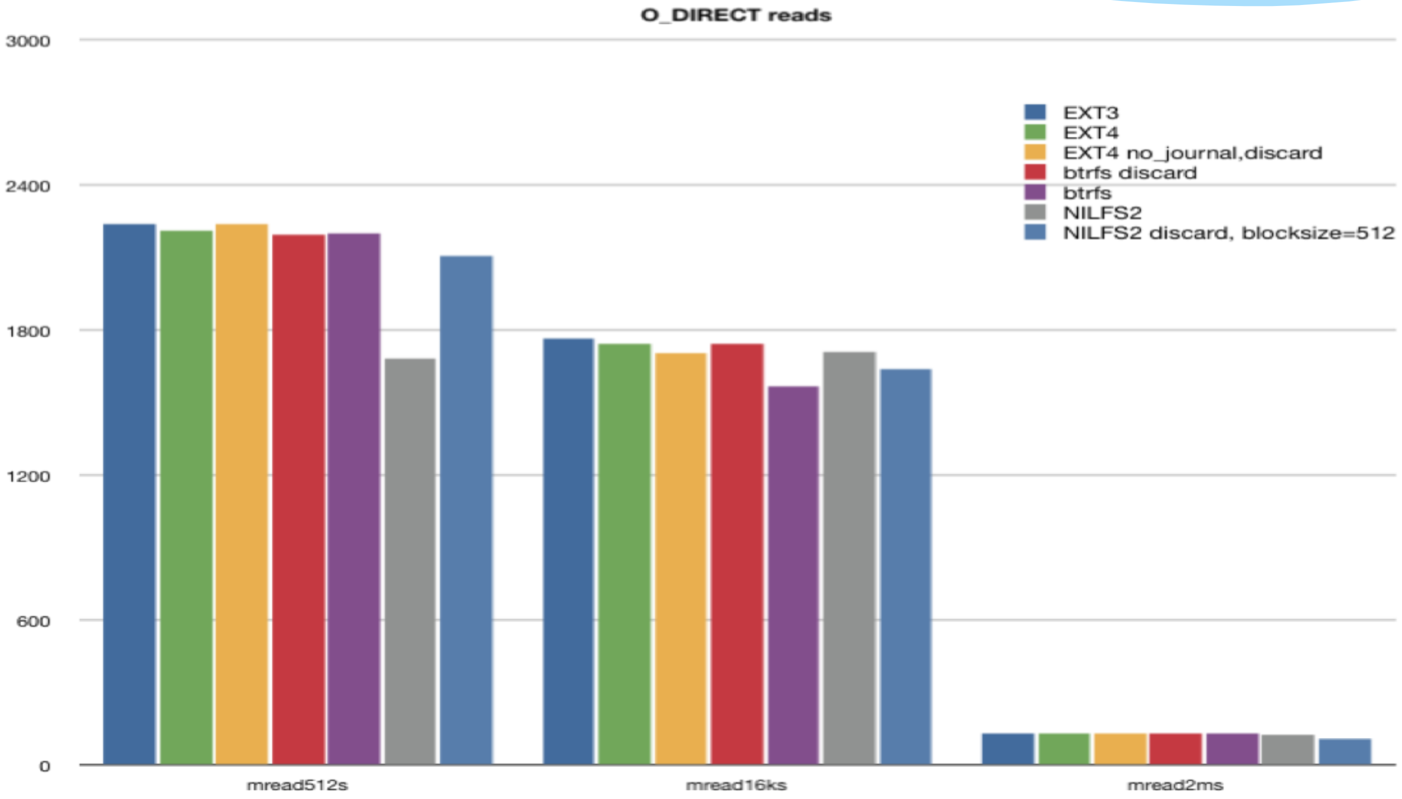
Filesystem Benchmarks




16KB writes
400
300
200
100
0
EXT3
EXT4
EXT4 no_journal,discard
btrfs discard
btrfs
NILFS2
NILFS2 discard, blocksize=512
mwrite16k
swrite16k
mwrite16ks
swrite16ks


512 byte writes
3000
2250
1500
750
0
EXT3
EXT4
EXT4 no_journal,discard
btrfs discard
btrfs
NILFS2
NILFS2 discard, blocksize=512
mwrite512
swrite512
mwrite512s
swrite512s


Synthetic workloads
2000
1500
1000
500
0
EXT3
EXT4
EXT4 no_journal,discard
btrfs discard
btrfs
NILFS2
NILFS2 discard, blocksize=512
varmail
fileserver
webproxy
androidload


O_DIRECT reads
3000
2400
1800
1200
600
0
EXT3
EXT4
EXT4 no_journal,discard
btrfs discard
btrfs
NILFS2
NILFS2 discard, blocksize=512
mread512s
mread16ks
mread2ms

EXT4 - a write

- Journal write (usually ~16K)
- inode update (usually 4K)
- Data goes into page cache

BTRFS - a write

- Update non-sync very fast
- Sync write puts tree leaves on eMMC
- Sync write is 4 non-sequential writes

NILFS2 - a write

- Log structured filesystem
- Stores the ‘update’
- One large (40K+) write
- Eventually “snapshot” needs flushing
- Initialization
- Recovery

EXT4 w/o journal

- Not too dangerous on embedded systems with battery
- Good performance due to improved sequentiality

BTRFS

- If not using a lot of fsync/fdatasync
- Great large write performance
- Terrible on small/medium sync writes
- Good performance on multiple writes

NILFS2

- Consistent performance
- Potentially much faster if eMMC part has fast sequential performance
- Should theoretically be the fastest :-)

EXT4 with journal

- If journaling is needed, consider RAM journal device
- Again RAM journal not as dangerous as you think
- Better than BTRFS on small/medium sync writes

I/O schedulers

- CFQ, noop, deadline
- Results are similar within ~10% range
- QOS considerations are more important than throughput

Filesystem layout

- No swap
- Align partitions to erase block boundaries
- Extents match erase blocks
- System design (multiple storage devices)

User space

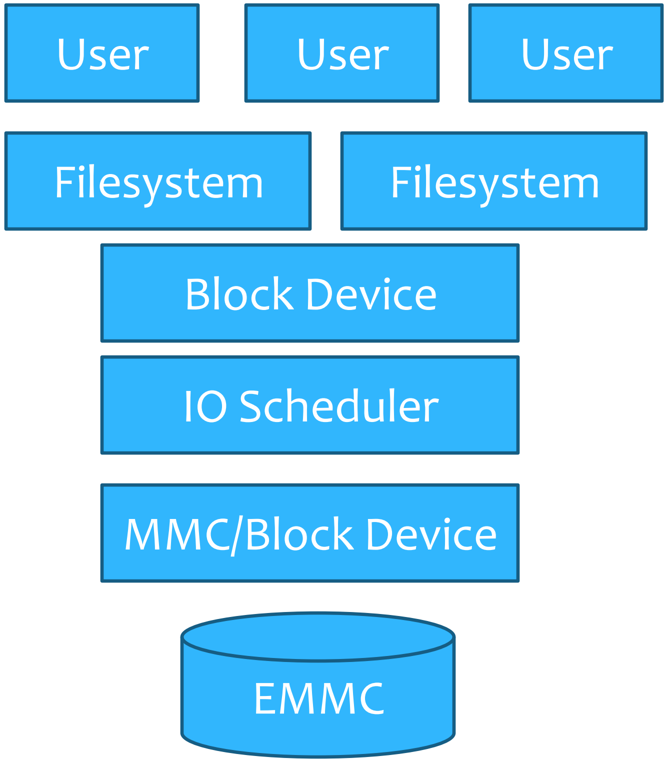
- Avoid synchronization on files
- Avoid sync/fsync/fdatasync/etc
- Avoid small writes to files, better to buffer
- Don’t be afraid to read, be afraid to write!

Future

- Linaro project (www.linaro.org) working on improving eMMC experience
- eMMC 4.5 brings METADATA

Summary

- User space
- Filesystem type
- Filesystem layout
- IO Scheduler
- Block IO & Cache
- MMC bus driver



Conclusion

- EXT4 (discard, ram/no journal) is probably your best bet
- Try out a couple of configurations for the eMMC you are targeting
- Benchmark per Vendor
- Avoid writes! :-)

Questions?