A Novel I/O Scheduler for SSD with Improved Performance and Lifetime

HuaWang[†], ***Ping Huang***[†‡], *Shuang He*[†], *Ke Zhou*[†✉], *Chunhua Li*[†], *and Xubin He*[‡]

[†] *Wuhan National Laboratory for Optoelectronics, HuaZhong University of Science and Technology, China*

[‡] *Department of Electrical and Computer Engineering,Virginia Commonwealth University, U.S.A*

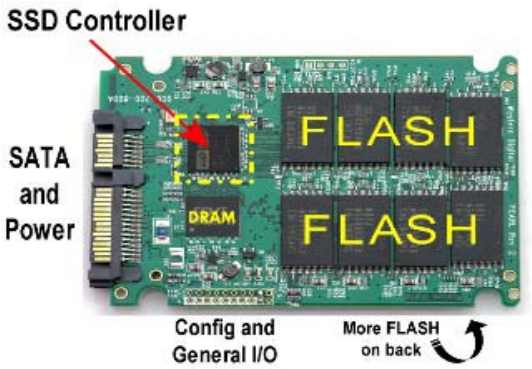
Presented at MSST, 2013-5-9

Outline

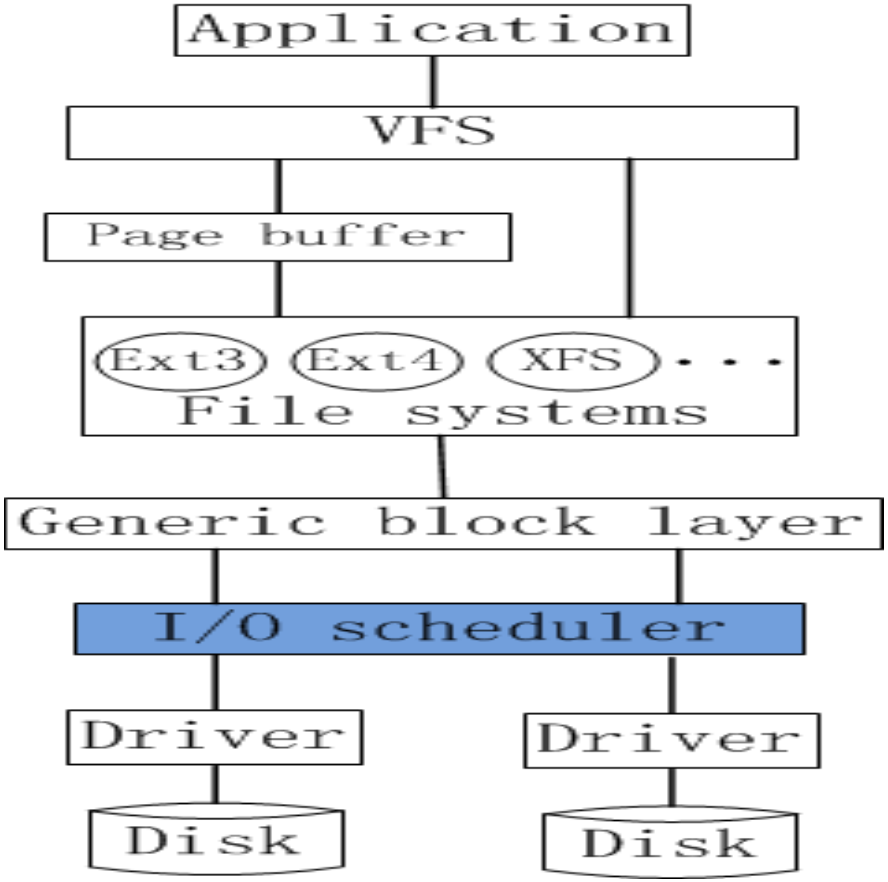
- Background
  - Features of Flash-based SSDs
  - I/O Scheduler Component
- System Design and Implementation
  - Overall Architecture
  - Techniques Deployed
- Evaluation Results
  - Evaluation Setup
  - Performance Report
- Conclusion

Features of Flash-based SSD

- Non-volatile Storage Media
- No Mechanical Components
- High Performance
- Read/Write Asymmetry
- Low Power Consumption
- Small Size
- Limited Lifetime
- Out-of-place Update
- Rich Internal Parallelism



The IO Scheduler



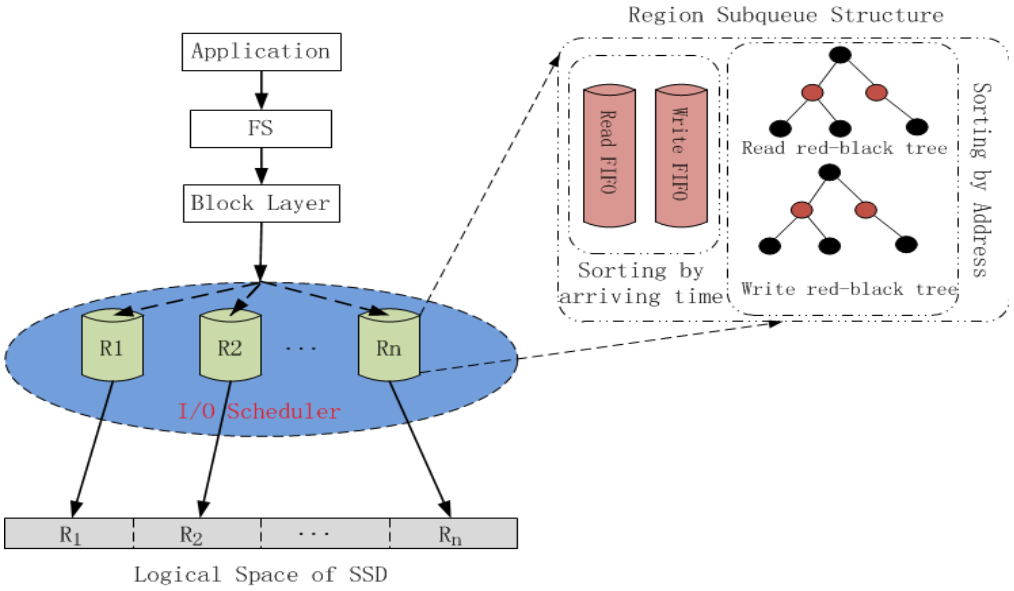
The Linux Off-the-shelf Scheduler

- Noop: it does not perform much optimization, rather it only checks whether to merge adjacently arriving requests
- Deadline: assign a dispatching deadline to each of the coming request to guarantee responsiveness and avoid starvation; merge and sort the waiting requests by address to reduce seeking latency
- CFQ: it tries to allocate disk resource among the competing processes fairly in a round-robin
- AS: the scheduler waits a brief period of time for anticipated requests instead of switching to serve other requests right away to attack "deceptive idleness"

However, except noop, they are all HDD-oriented, with seeking latency as the biggest concern

System Design and Implementation

- Our Approach: instead of passively using "Noop" scheduler, we actively dispatch requests to the SSD in parallel to leverage the rich parallelism existing within SSDs.
- Overall Architecture:



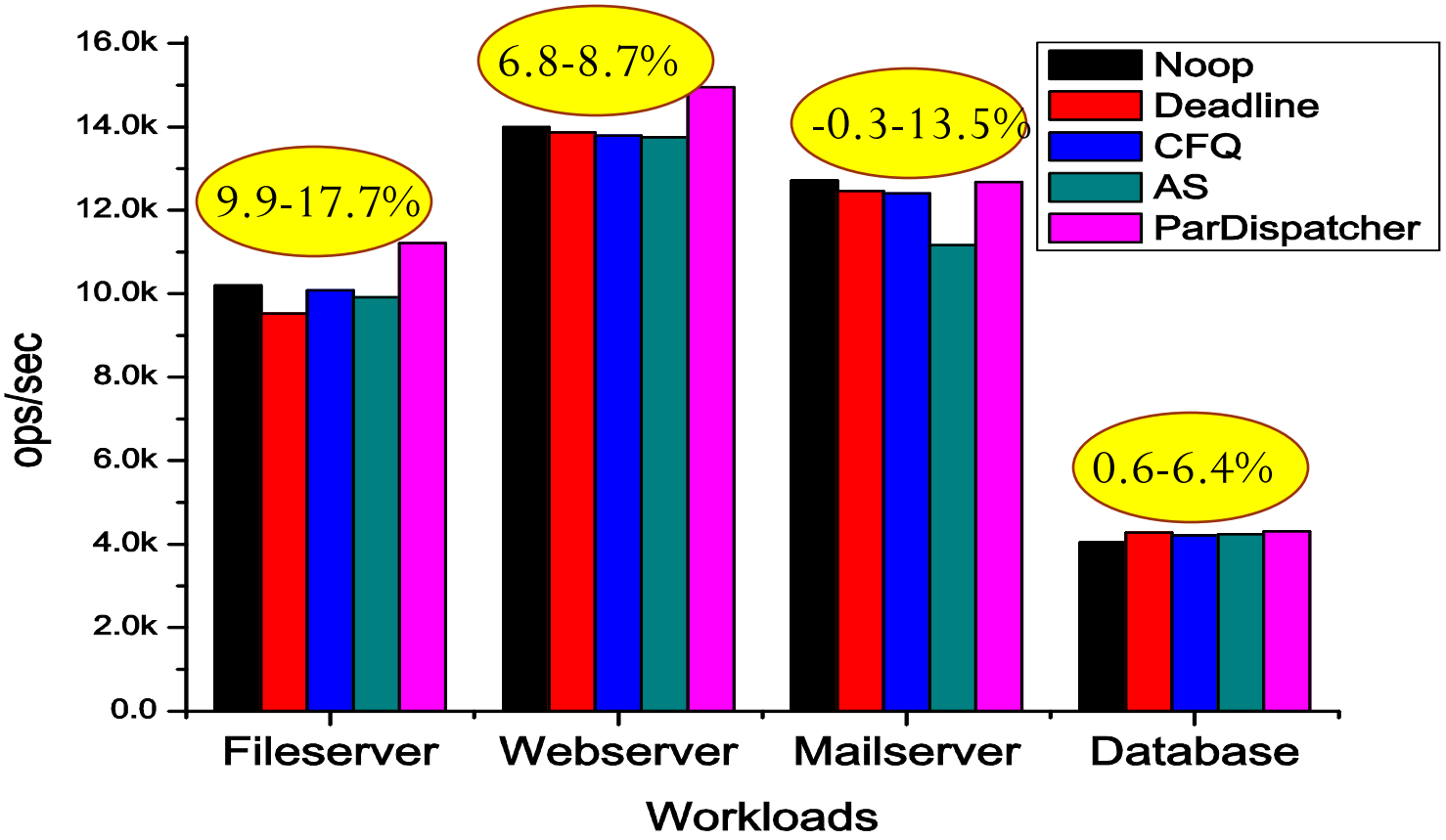
Deployed Techniques

- Space Partition: the entire space is divided into many different regions according to logical addresses and requests are tracked in respective regions according to their accessing addresses
- Request Sorting: requests visiting to the same region are sorted as well in order to leverage better sequential performance
- Interference Avoidance: for each dispatching chance, the scheduler only dispatches a batch of read or write requests alternatively to reduce the severe interference problem

Evaluation Setup

- Implemented as a kernel module consisting of about 1000 line of codes(LOC) against Linux kernel version 2.6.32
- Kingston MLC 60GB SSD as our testbed
- Use FileBench to generate four representative workloads to demonstrate the scheduler's advantages
- Use blktrace to collect block traces
- Use those collected traces to drive Flashsim with various FTL schemes to demonstrate its lifetime friendliness

Performance Comparison



Lifetime Improvement

	Noop	Deadline	CFQ	AS	ParDispatcher
DFTL	32102/34	31543/32.7	31160/32	30356/30	21214
PM	30244/35.8	29717/34.7	29356/34	28598/32.2	19401
FAST	306957/45.3	298075/43.6	248418/32.4	277993/39.6	167953

The experienced erase operations of Webserver under different FTLs. Each cell gives the number of erase operations and the saved percentages relative to ParDispatcher

Conclusion

- We design an SSD-oriented block I/O scheduler which proactively leverages the internal parallelism
- We demonstrate its effectiveness in improving performance and lifetime with a variety of workloads in respect to the four off-the-shelf schedulers
- We plan to compare it with other SSD-oriented schedulers, e.g., FIO(FAST'2012) scheduler

Thank you!