A Machine Learning Framework to Improve Storage System Performance

13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2021)

Ibrahim "Umit" Akgun, Ali Selman Aydin,

Aadil Shaikh, Lukas Velikov, and Erez Zadok

Stony Brook University

Tunable Parameters in Systems

- Thousands of tunable parameters in the Linux kernel
- Examples
  - Readahead size (# sectors)
  - I/O scheduler (kyber, mq-deadline, bfq)
- Why are they important?
  - Affect performance and latency
  - Depends on workload
  - Especially for I/O subsystems

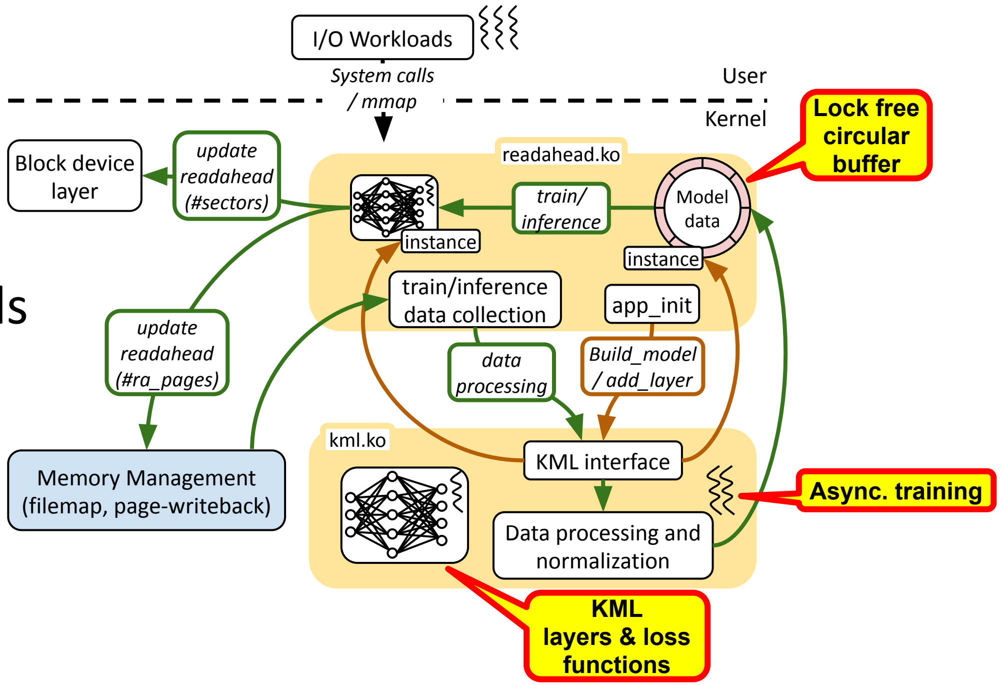
[Cao et al. FAST'17,
Sehgal et al. FAST'10]

Adjusting Kernel Tunable Parameters

- Why can’t we tune them manually?
  - ❖ Highly depend on workloads
- How frequently need to tune these parameters?
  - ❖ Must detect workload changes dynamically
- Why is the tuning so complicated?
  - ❖ Expertise: need to know impact of many parameters
  - ❖ Inter-dependencies **[Cao et al. FAST’17]**
- Our approach
  - ❖ KML framework: Machine Learning Framework for OSs and Storage Systems

KML Design Principles

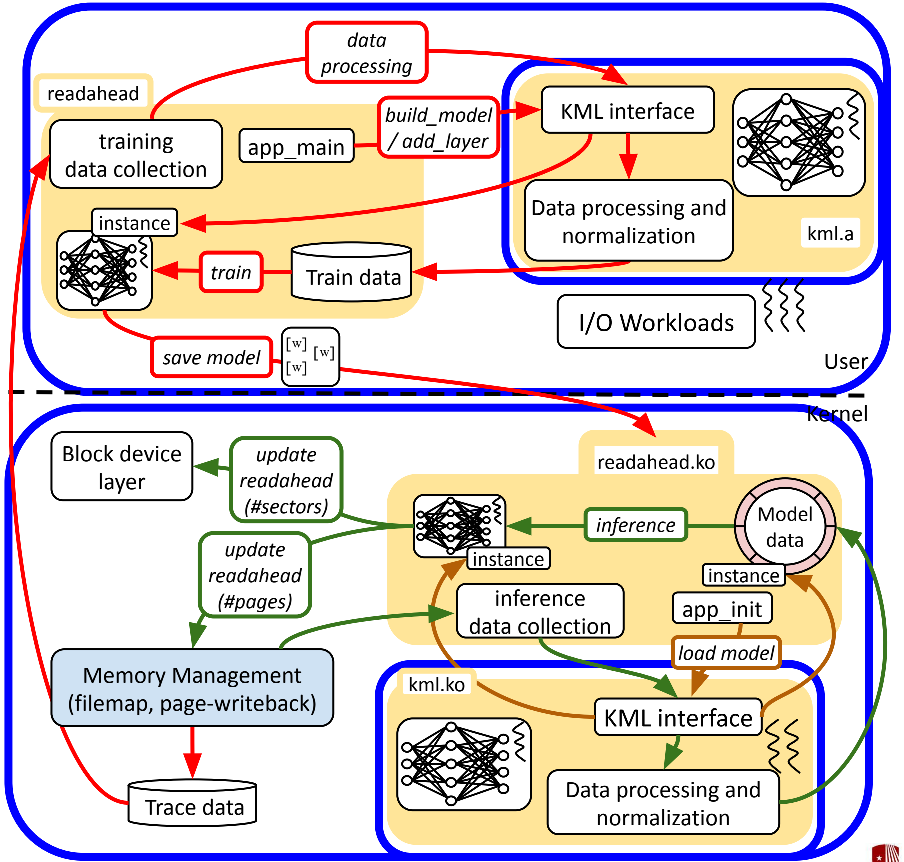
- Library Design
  - Math and matrix operations
  - Layers and loss functions
  - Inference and training (DAG)
  - Automatic differentiation
- Reducing computational overheads
  - Floating-point operations & context-switches
- Reducing memory overheads
  - Asynchronous training
  - Lock-free circular buffers for data collection


I/O Workloads
System calls / mmap
User
Kernel
Lock free circular buffer
Block device layer
update readahead (#sectors)
readahead.ko
train/ inference
Model data
instance
instance
train/inference data collection
app_init
update readahead (#ra_pages)
data processing
Build_model / add_layer
Memory Management (filemap, page-writeback)
kml.ko
KML interface
Async. training
Data processing and normalization
KML layers & loss functions

Stony Brook University

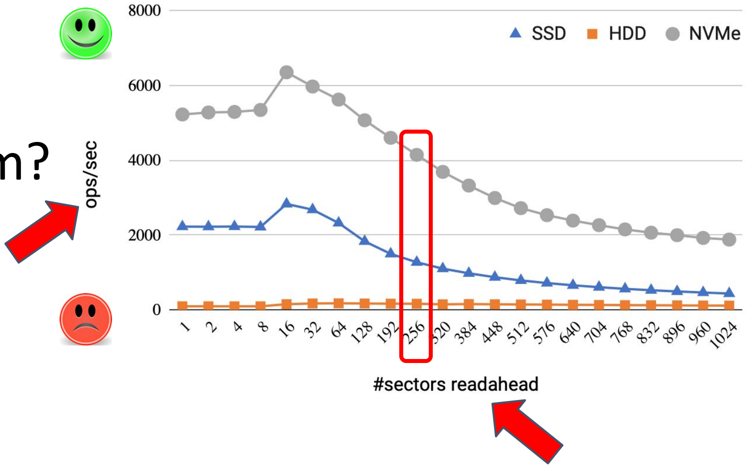
KML API

- Can generate model to both user and kernel space
- User space training & kernel space inference
- Kernel space training and inference
  - Reinforcement learning (future work)
  - More efficient data collection in kernel
- KML's programming model
  - Versatility
  - Safety
  - Low-overhead



Use Case: Readahead (Motivation)

- What is the readahead problem?
  - Tune with fadvise and madvise
- How did we study the readahead problem?
  - RocksDB on SSD & NVMe, four workloads
  - 22 different readahead sizes (8 – 1024)
- Data Collection
  - Tracepoints via LTTng (add_to_page_cache, writeback_dirty_page)
  - mmap event extensions [Re-Animator, SYSTOR’20]
  - Tracing overheads

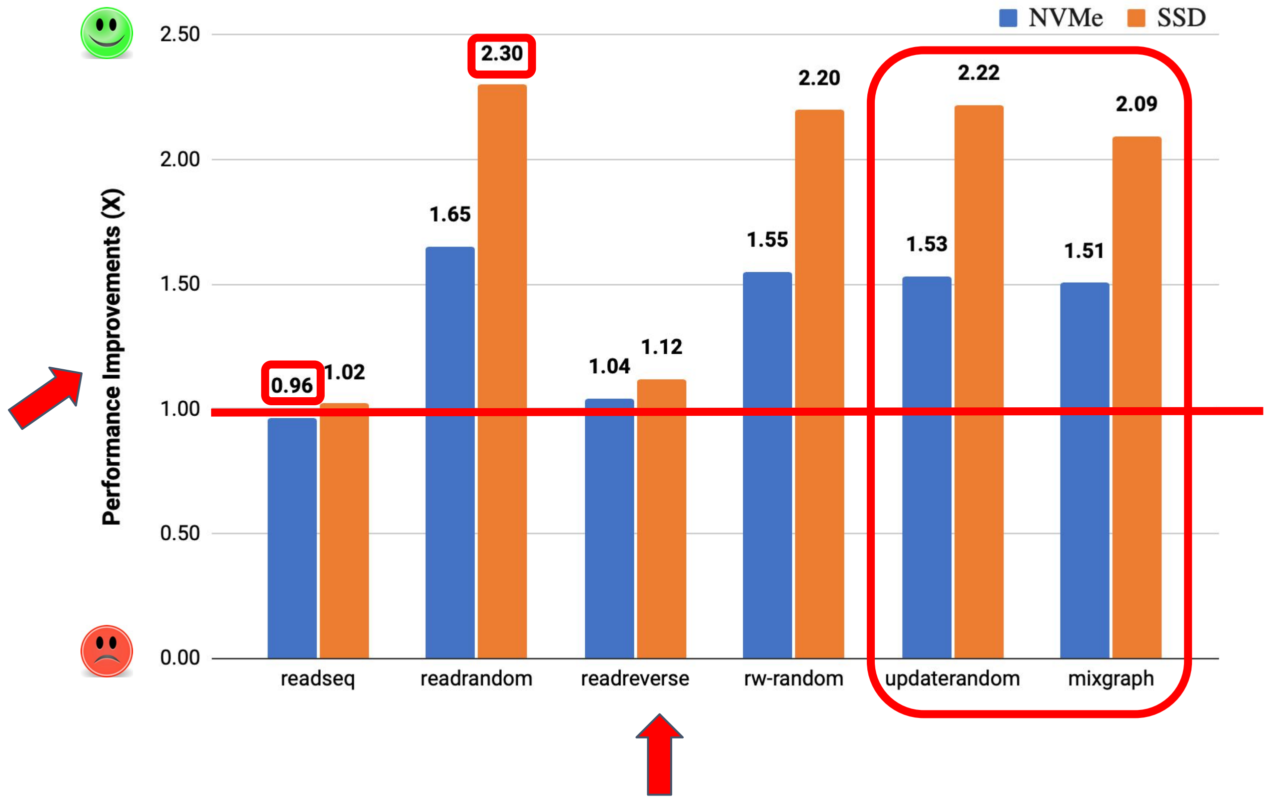


Stony Brook University

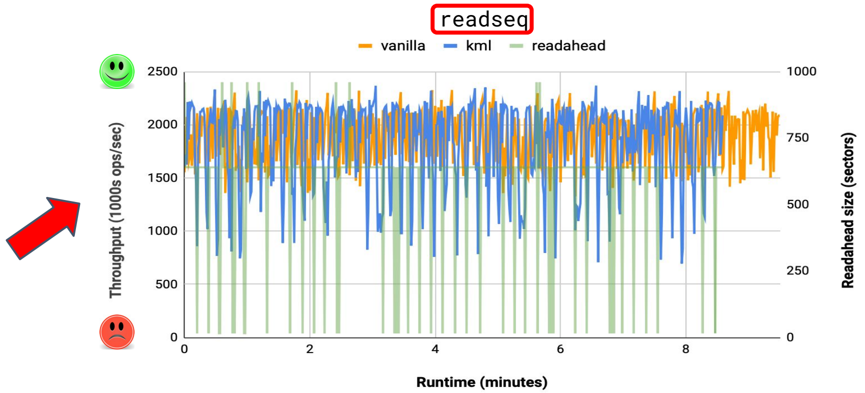
Use Case: Readahead (Implementation)

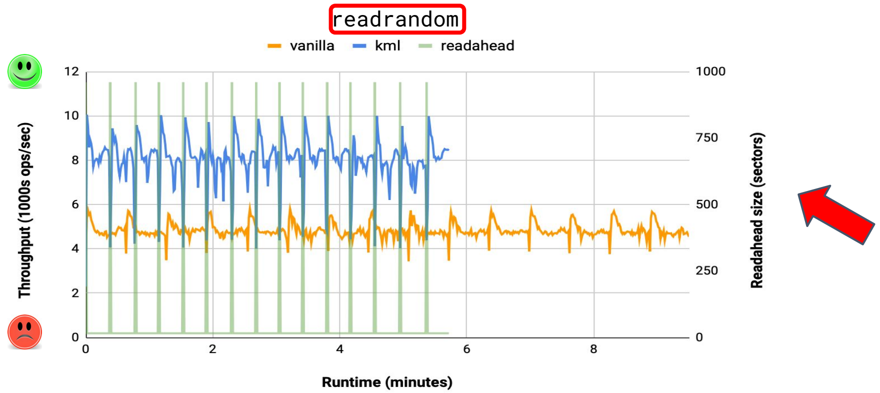
- Data pre-processing & feature extraction
  - Features
    1. Number of tracepoints that were traced (transactions)
    2. Cumulative moving average of page offsets
    3. Cumulative moving standard deviation of page offsets
    4. Mean absolute page offset differences for consecutive tracepoints
    5. Current readahead value
  - Pearson correlation, k-fold validation
- Neural network model
  - Multi-class classification
  - Trained on only readrandom, readseq, readreverse, and rw-random on NVMe
  - Three linear layers connected with sigmoid activation function
  - 10-fold validation: average accuracy 95.5%

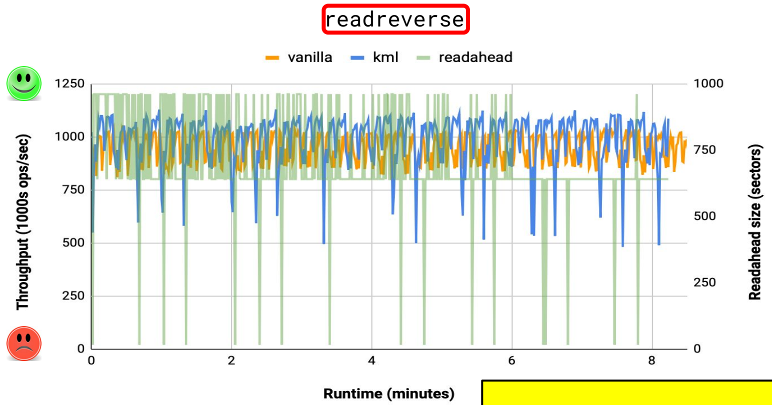
Evaluation: Performance Improvements

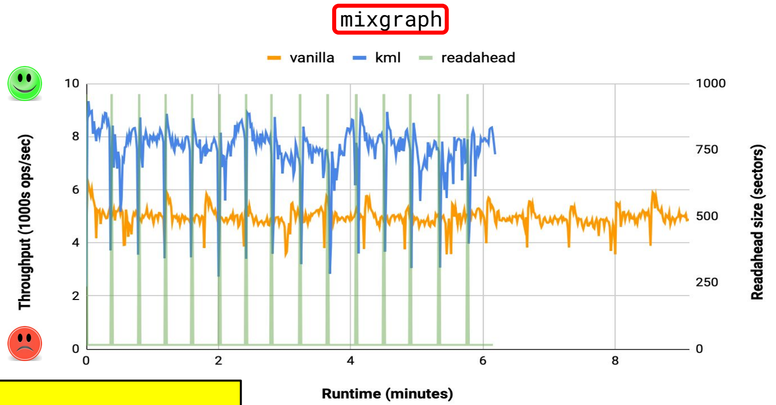


Evaluation: Adaptability









Overall 30% better throughput

Stony Brook University

Conclusion & Future Work

- KML can improve RocksDB I/O performance up to 2.3x
- Quickly adapts to changing workloads
- Apply to other storage subsystems and OS problems
  - I/O schedulers
  - NFS
  - Page cache
- Extend KML framework
  - Reinforcement Learning
  - Computing DAGs with multiple threads
  - Support arbitrary computation DAGs

A Machine Learning Framework to Improve Storage System Performance

Q&A

13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage 2021)

Ibrahim "Umit" Akgun, Ali Selman Aydin,

Aadil Shaikh, Lukas Velikov, and Erez Zadok

Stony Brook University