Online Disk I/O Analysis

By Michael Marzullo

Introduction

Introduction

- Performance of storage devices has increased dramatically in recent years and moved to newer technologies

- Comparatively, not many advances have been made in software

- There are many opportunities to get the maximum performance out of the hardware available

Introduction

- Data mining of frequent itemsets can be used to generate block correlations that can be then used for optimization
- Two categories
  - Offline
  - Online

Related Work

- Apriori
- ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal)
- C-Miner
- estDec+
- Adaptive Replacement Cache

Monitoring Tools

- Blktrace - Linux block device monitoring tool
- Blkparse - Parses binary output of blktrace

- These tools have limitations – can only be used for offline analysis

Experimentation Tools

- Fio
- Filebench
- Blkreplay

- Cannot generate specific overwrites of blocks using these benchmarking tools
  - Custom tooling is needed

Implementation

Linux Kernel/Userlevel applications

- Need monitoring of specific process identifiers
- Blktrace has both a kernel and userlevel component
  - Both require modifications

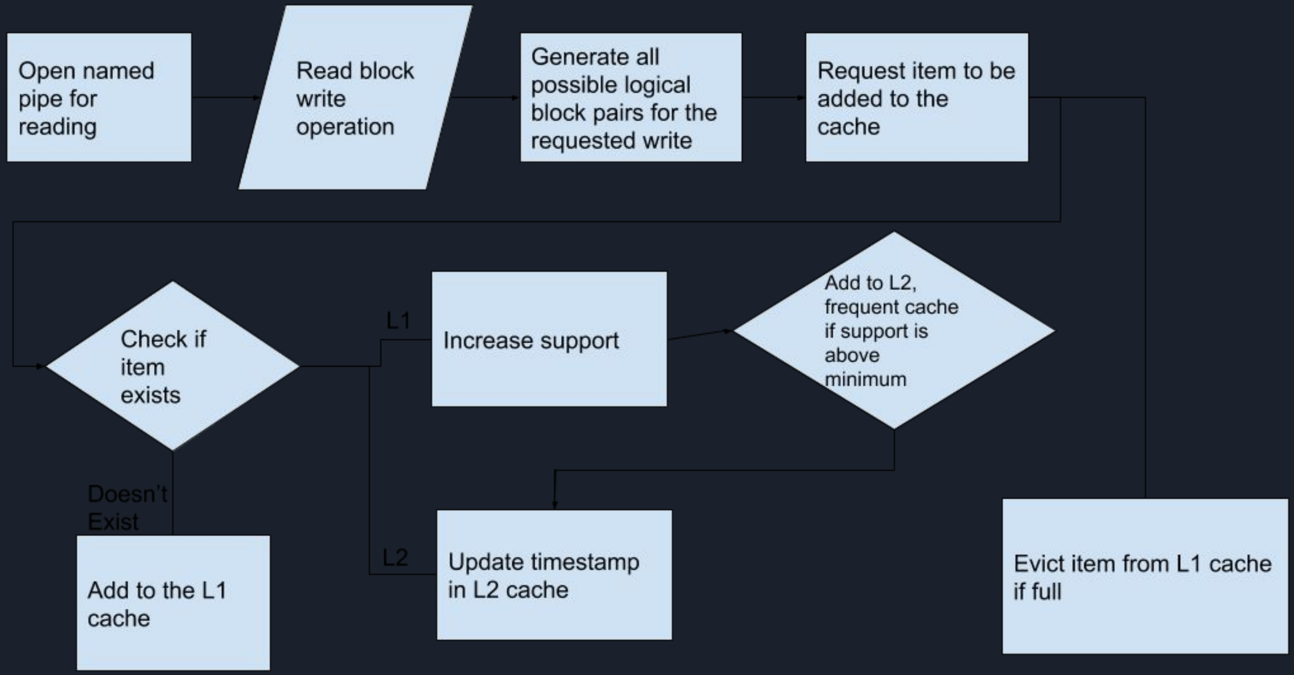
- Add filtering parameter
  - We are only interested in writes from specific processes
- Add functionality to analyze the I/O in real-time

- Custom kernel is needed

Algorithm 1

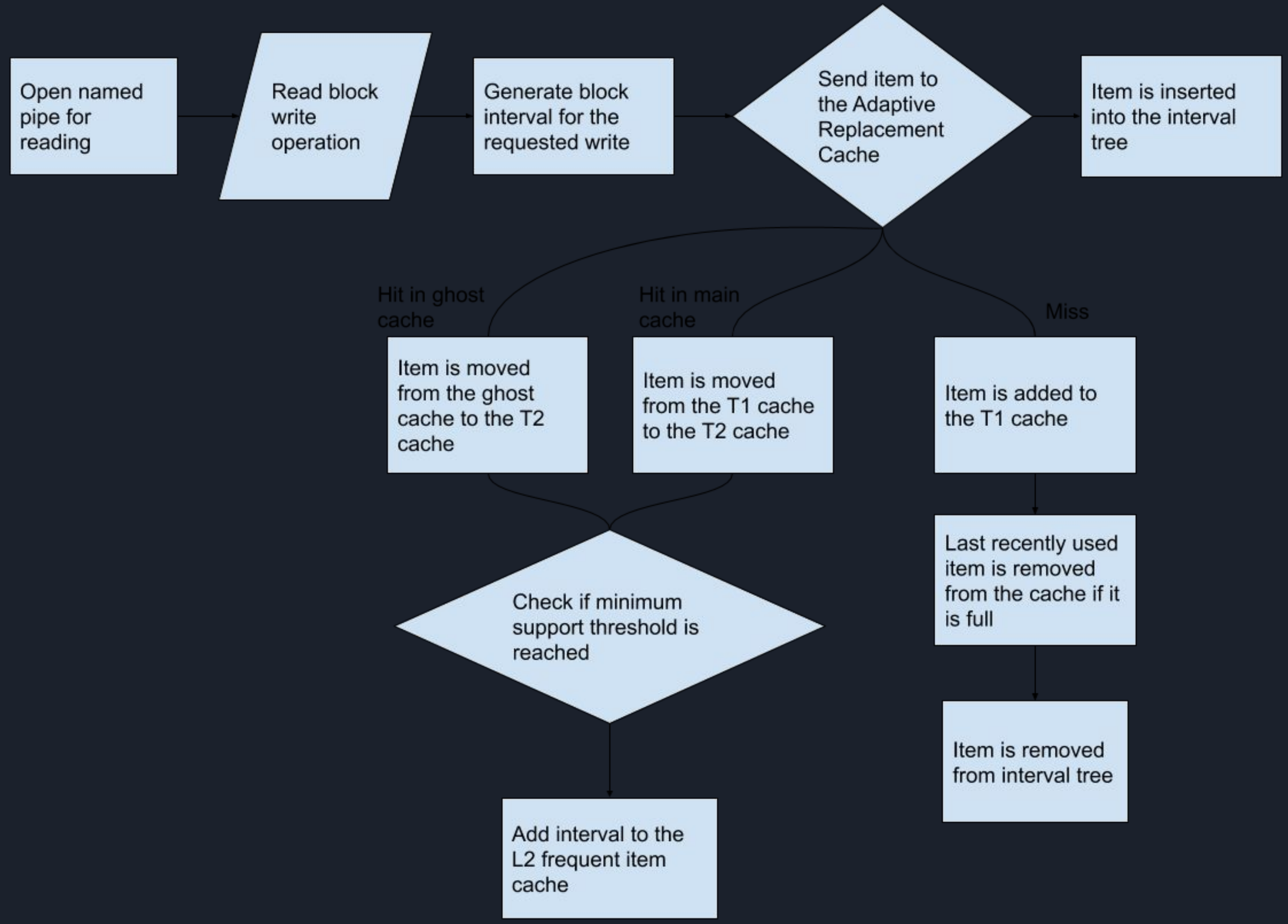
- 3 Level cache
- Entry cache
- Frequent item cache
- Eviction cache

- Input: all possible block pairs for each incoming I/O request
    - Computationally Expensive O(N^2)


Open named pipe for reading
Read block write operation
Generate all possible logical block pairs for the requested write
Request item to be added to the cache
Check if item exists
L1
Increase support
Add to L2, frequent cache if support is above minimum
Doesn't Exist
L2
Add to the L1 cache
Update timestamp in L2 cache
Evict item from L1 cache if full

Algorithm 2

- Uses ARC and interval tree instead of generating all block pairs
- Still multi-level
    - Has a simple LRU frequent item cache + interval tree


Open named pipe for reading
Read block write operation
Generate block interval for the requested write
Send item to the Adaptive Replacement Cache
Item is inserted into the interval tree
Hit in ghost cache
Hit in main cache
Miss
Item is moved from the ghost cache to the T2 cache
Item is moved from the T1 cache to the T2 cache
Item is added to the T1 cache
Last recently used item is removed from the cache if it is full
Check if minimum support threshold is reached
Item is removed from interval tree
Add interval to the L2 frequent item cache

Experimental Testbed

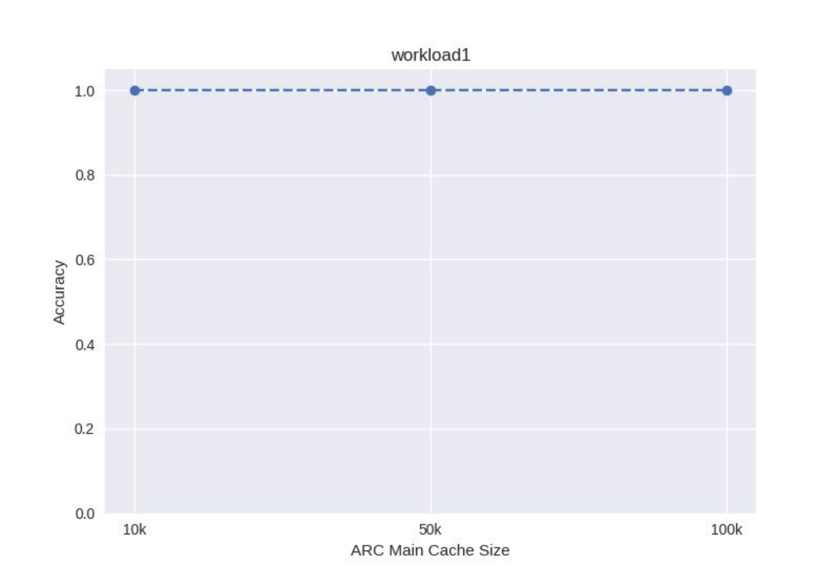
- Intel Xeon E3-1230v5
- 64GB RAM
- Samsung PM961
- 10 GB Block Device
- 512B Sector Size (20971520 total sectors)

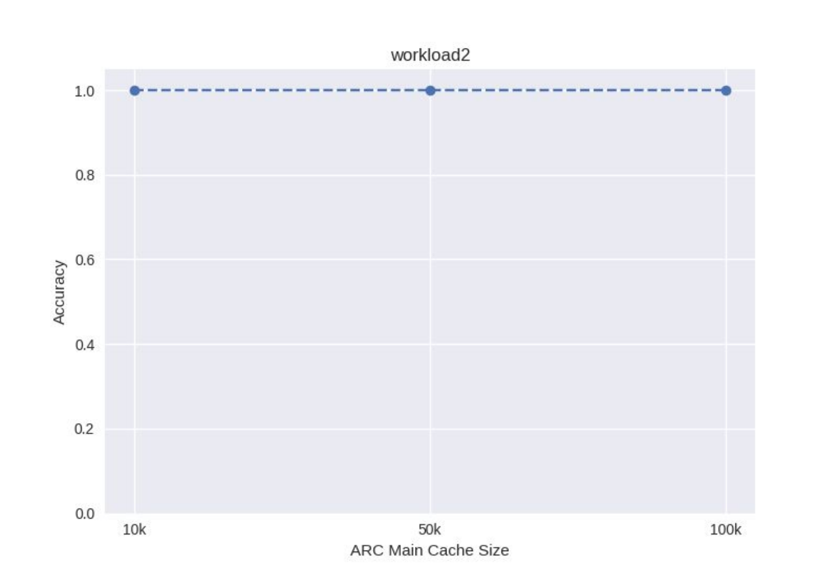
Experiments

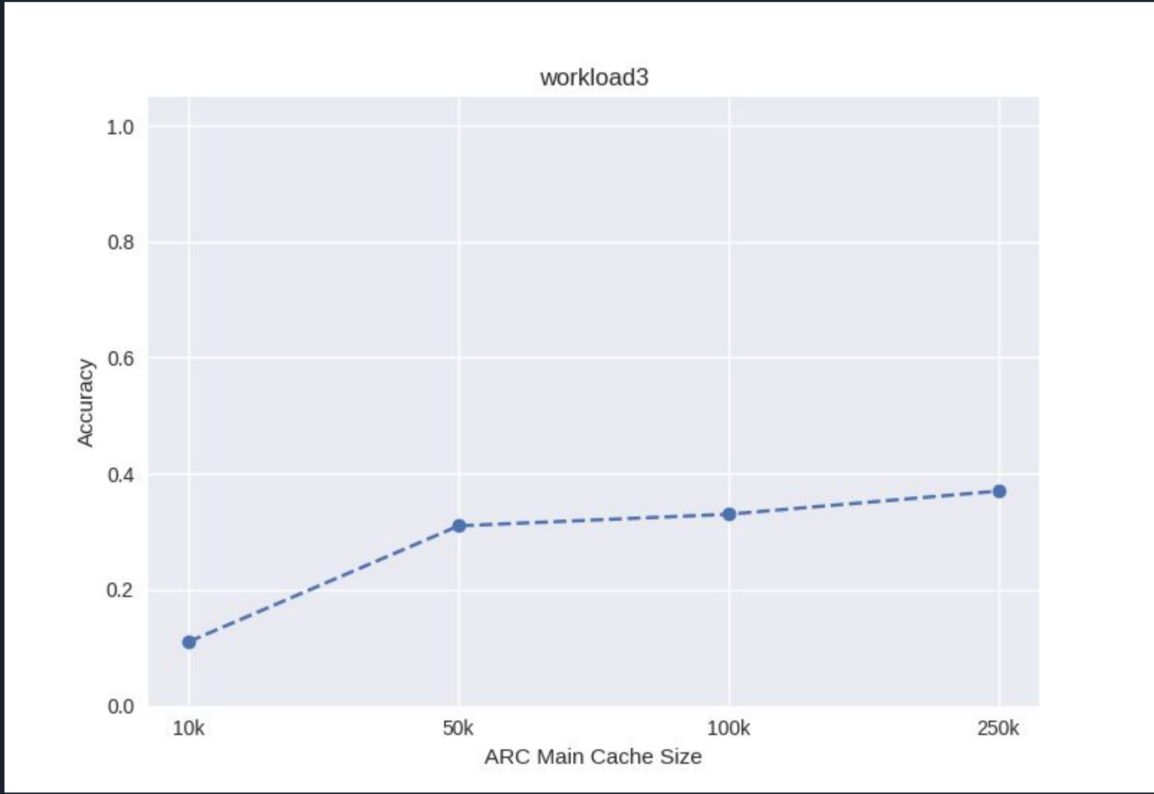
Name	Size	Minimum Support	Sector Patterns	Pattern Probability
Workload1	1m	30	100	0.1%
Workload2	1m	30	100	0.01%
Workload3	1m	10	100	0.001%
Workload3ms5	1m	5	100	0.001%

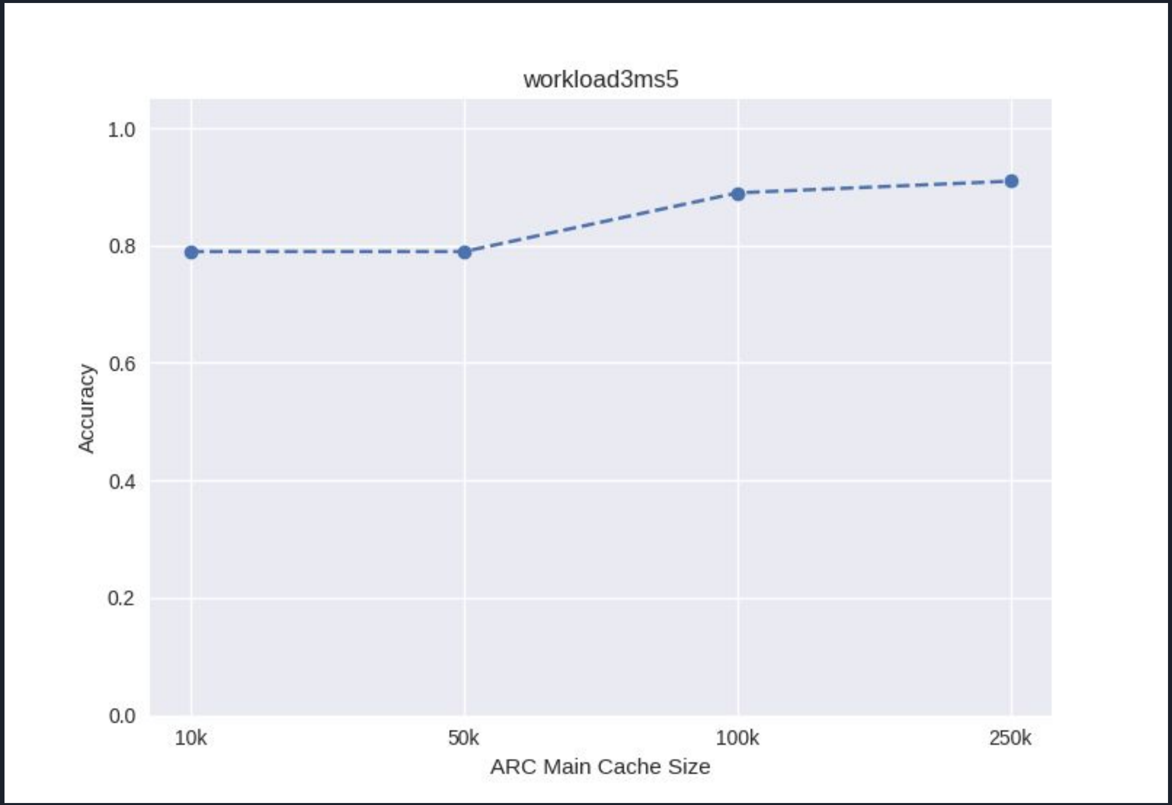
Results

Tracing Performance
Tracing Disabled
Tracing Enabled
175000
150000
125000
100000
75000
50000
25000
0
Bandwidth (KB/s)
IOPS


workload1
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
10k
50k
100k
ARC Main Cache Size


workload2
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
10k
50k
100k
ARC Main Cache Size


workload3
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
10k
50k
100k
250k
ARC Main Cache Size


workload3ms5
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
10k
50k
100k
250k
ARC Main Cache Size

Conclusions

- Algorithm works successfully on synthetic traces, fulfills original goals
- Fast, accurate
- Still many areas for improvement

Future Work

- More experiments
  - Synthetic: different distributions
  - Real traces
  - Simulations of hardware (open-channel SSD)
- Optimize algorithm
  - Convert to higher performance/systems level language
  - New eviction policies
  - Automatic parameter tuning
- Tracing mechanism
  - Support asynchronous I/O