SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

Data Placement

Marc Acosta, Research Fellow
Office of the CTO
Western Digital

Data Placement

- Imagine this bag is your storage system and the different color beads are your files
  - Then imagine the white beads need to be moved to another storage system

https://www.amazon.com/Pony-Beads-Multi-Color-1000/dp/B004D9DMMW/ref=psdc_12896121_t1_B07PC14Q4J

Data Placement

- Now Imagine this is your storage system and the different color balls are your files
  - Did moving the white beads just get more efficient

https://www.amazon.com/Efivs-Arts-Multicolor-Beading-Container/dp/B07T3HL8T4

Data Placement

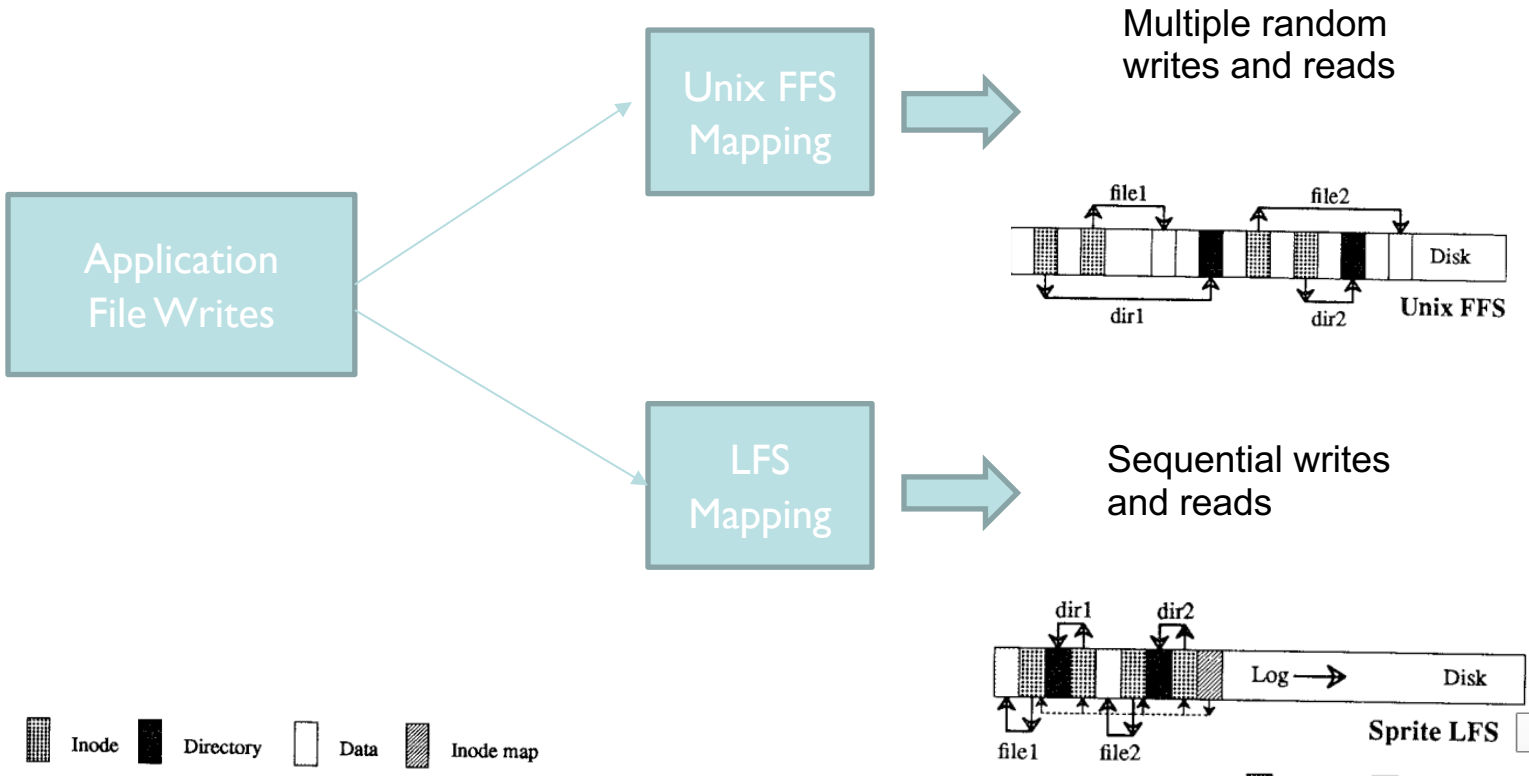
- Storage is really about two things
  - Initial data placement and changes to data
- Storage medias tend to like bigger changes over smaller changes
  - Large block writes are more efficient than small block writes

Data Placement

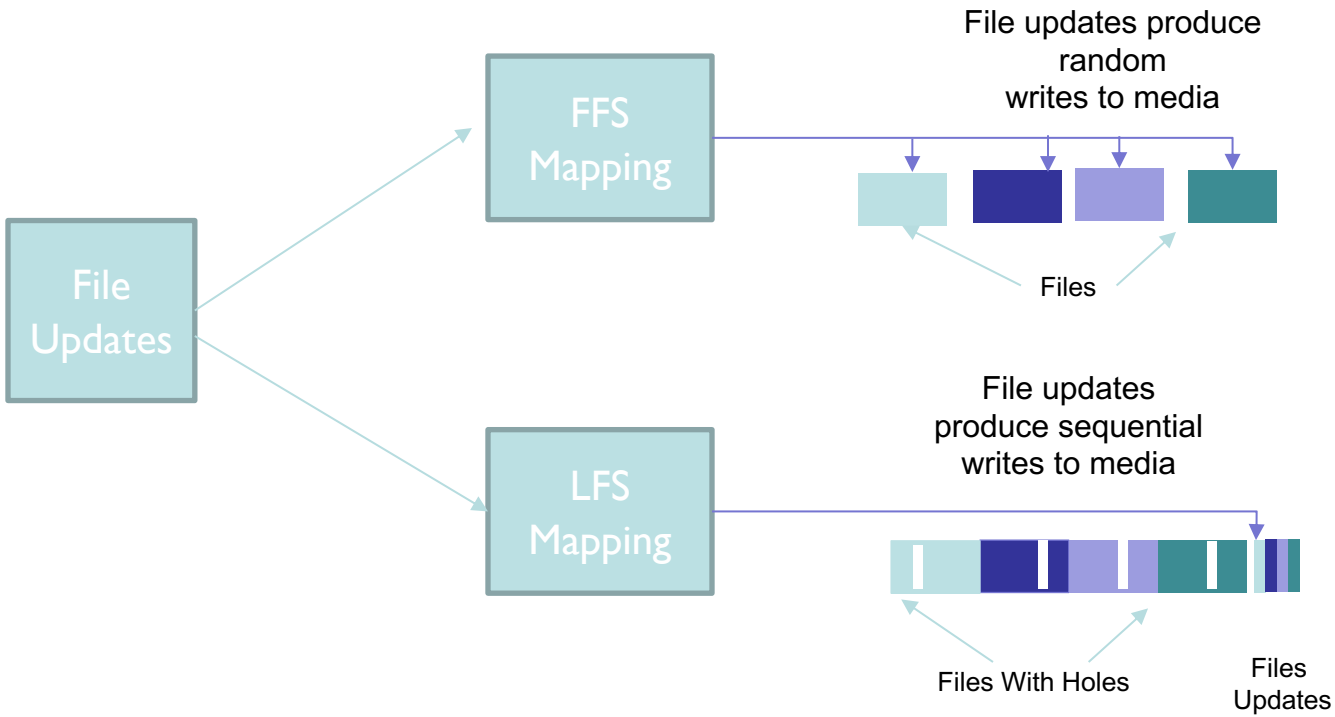
- Rosenblum and Ousterhout's Log Structured File System improved the file system by optimizing for the storage media
  - Their filesystem, Sprite LFS, reduced the mechanical movement of an HDDs heads by placing files sequentially on the media and placing file metadata close to the files
  - Sprite LFS managed updates to the files by sequentially writing changes to new place

https://web.stanford.edu/~ouster/cgi-bin/papers/lfs.pdf

Sprite LSF File Placement

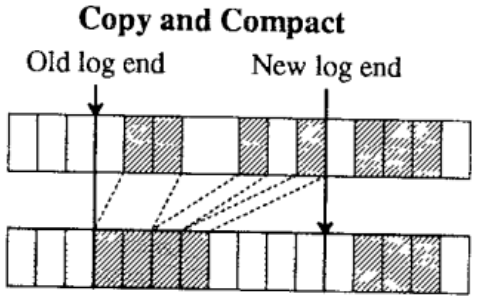


File Change: Sprite LSF Vs FFS



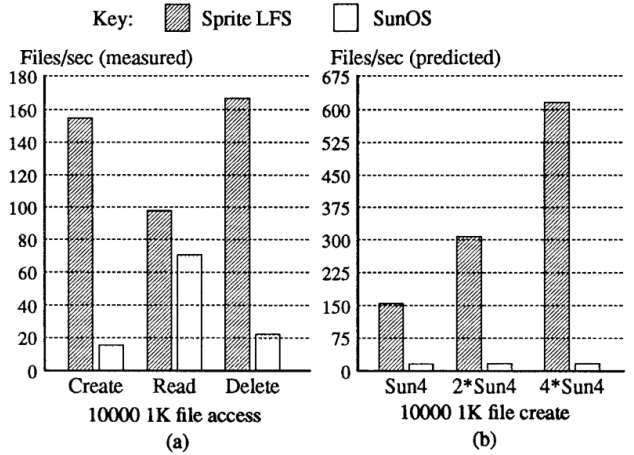
Sprite Copy and Compact

- Copy and Compact frees up larger segments by compacting smaller data segments together
  - Today this is known as garbage collection



Small is good but larger is better

- LFS Conclusion:
  - “works very well for large-file … in particular … very large-file that are created and deleted in their entirety ”
    - i.e. Immutable files
- Files / Storage with no updates



simple policy based on cost and benefit. Although we developed a log-structured file system to support workloads with many small files, the approach also works very well for large-file accesses. In particular, there is essentially no cleaning overhead at all for very large files that are created and deleted in their entirety.

https://web.stanford.edu/~ouster/cgi-bin/papers/lfs.pdf

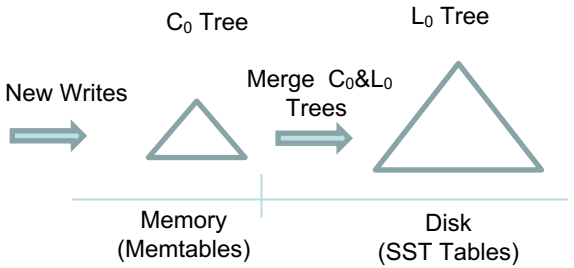
Log Structured Merge Tree

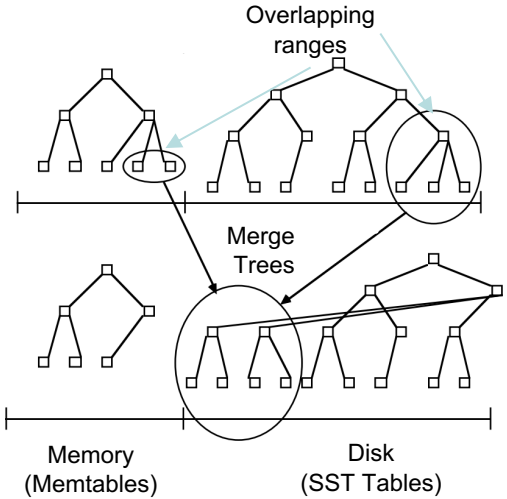
- Patrick O'Neil et al, Log Structured Merge Tree came across a similar change issue with B-tree indexes
  - Small changes to B-trees were inefficient
  - Big changes to B-trees were more efficient
- The LSMT algorithm defers and batches index changes to improve efficiency
  - Bulk loading of B-trees
    - See https://bigdata.uni-saarland.de/datenbankenlernen/ or Jens Dittrich's e-book Patterns in Data Management

https://link.springer.com/content/pdf/10.1007/s002360050048.pdf

Log Structured Merge-Tree (LSM-Tree)

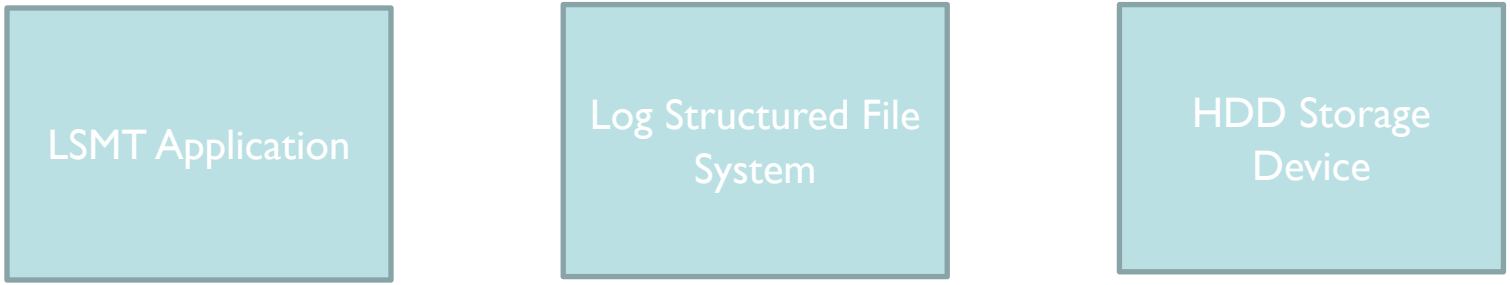
- LSMT does not modify data files
  - They are all immutable
- LSMT collects updates and writes out changes to a new file
  - To find a piece of data all branches of the tree need to be searched
- LSMT files are large immutable files





https:// www.cs.umb.edu/~poneil/lsmtree.pdf
The Log-Structured Merge-Tree (LSM-Tree) O'neil et al

End to End Data Placement



Minimizes GC LSF
-- Creates and uses immutable files

Minimizes Head Movement for higher B/W
-- Places data by file sequentially on media

Media is more efficient in sequential accesses
-- Maintains the spatial locality of the files on writes

End to End Data Placement

LSMT Application

Log Structured File System

Minimizes GC LSF
-- Creates and uses immutable files

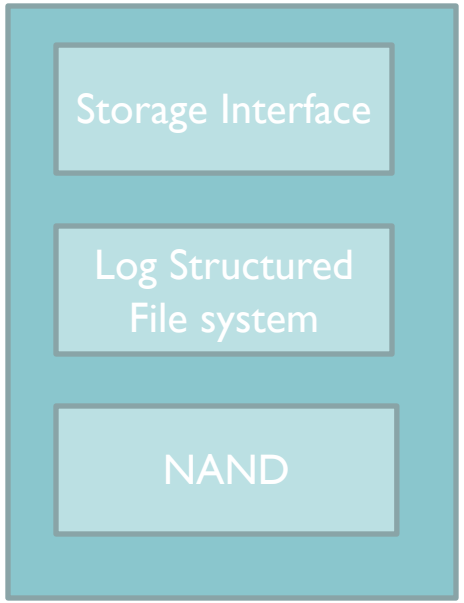
Minimizes Head Movement for higher B/W
-- Places data by file sequentially on media

Media is more efficient in sequential accesses
-- Maintains the spatial locality of the files on writes

Life with SSDs

NAND and SSDs

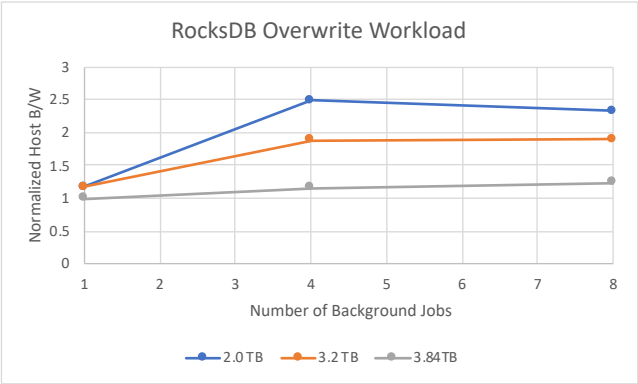
- NAND Flash Memory has an interesting property
  - NAND needs to be written and erased in large segments
- LSF writes in large segments and GC process creates empty segments
- Add a storage interface and you have created an SSD

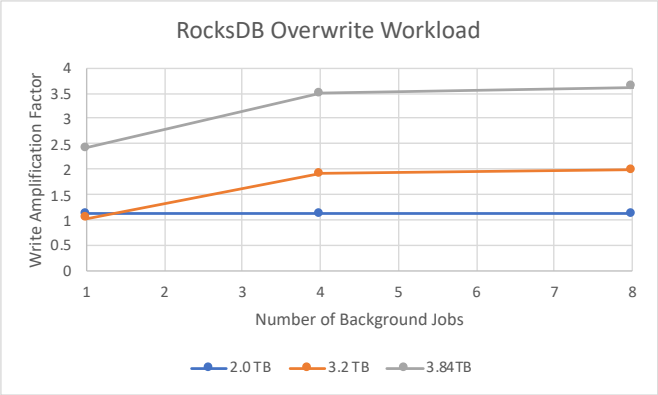


Modern SSD

LSMTs on SSDs

- SSDs do not show the same end to end efficiency
- RocksDB uses LSMT indexing
  - As the number of threads increased there was no scaling of performance
- Same workload as HDD but the SSD LFS introduced unnecessary write amplification into the system
- Performance scaling required overprovisioning
  - Reducing the user capacity of the drive

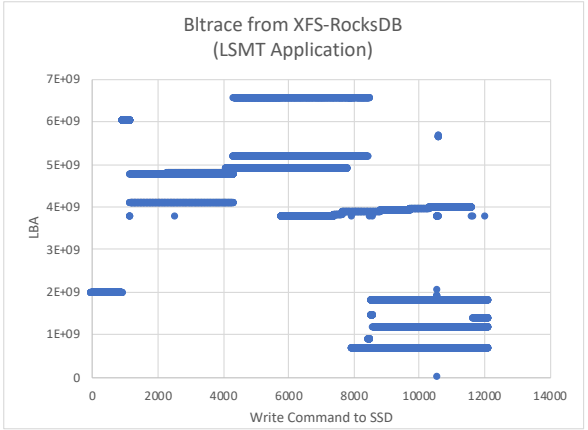




Blktrace of Workload

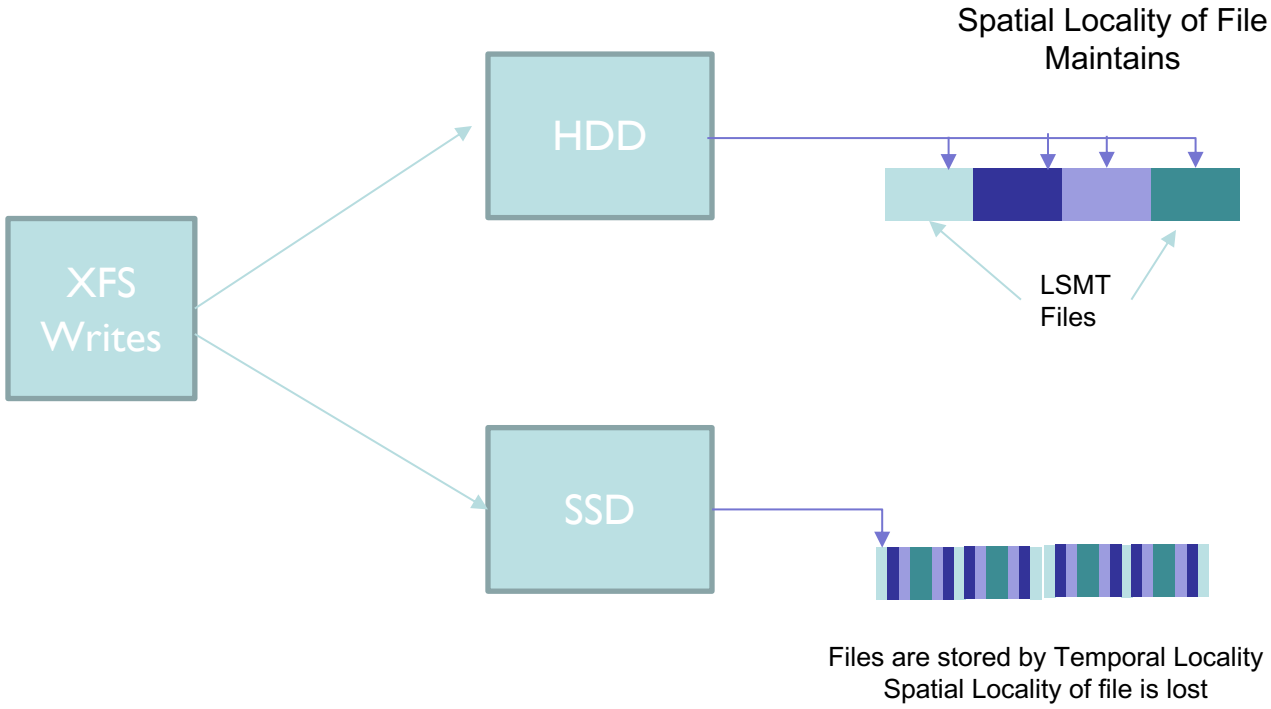
LSMT Application (RocksdB)

XFS (Extent File System)

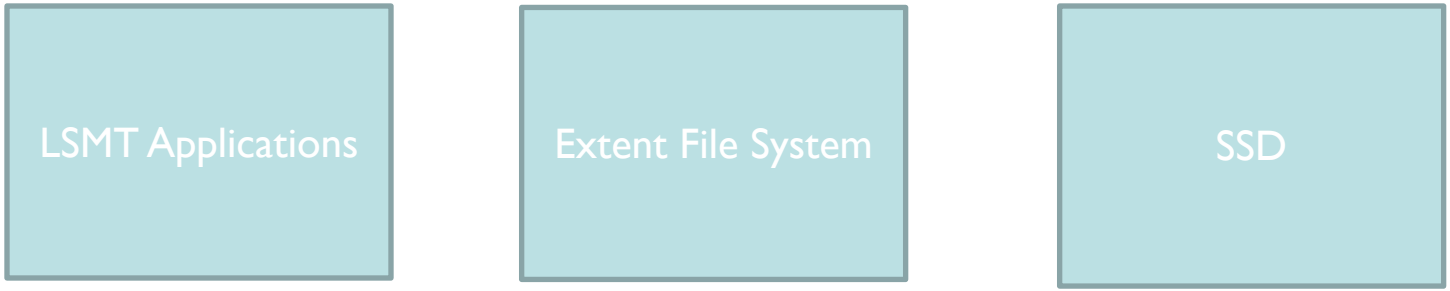


- The block addresses of files are allocated to place data sequentially on media

Sprite LSF Change



End to End Data Placement



Files are immutable

Places data by file sequentially in large blocks

Uses Temporal data placement
-- Does not place files sequentially on media
-- Does not take advantage of large immutable files

~~End to End Data Placement~~

LSMT Applications

Extent File System

Files are immutable

Places data by file sequentially in large blocks

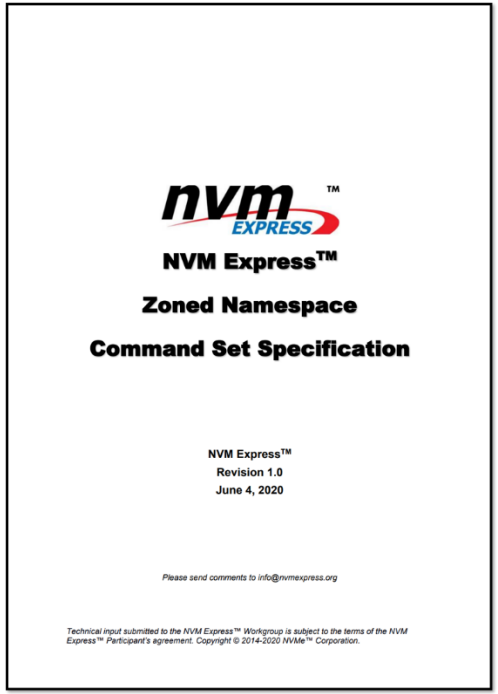
Uses Temporal data placement
-- Does not place files sequentially on media
-- Does not take advantage of large immutable files

How to fix

- Make an SSD that uses a block addressing method to place data spatially

Zoned Namespaces (ZNS) SSDs: Disrupting the Storage Industry

Matias Bjørling



https://nvmexpress.org/developers/nvme-specification/
(Available in the 1.4 TP package)

End to End Data Placement

LSMT Application

Log Structured File System

ZNS Storage Device

Optimizes Index Creation
Utilizes spatial locality of data to improve performance
Utilizes immutable files

Maintains Spatial locality of data files
immutable files reduce GC

ZNS addressing maintains spatial locality of data files
No GC

Fileolgy: Study of Files

A look into Cassandra Files

- Cassandra is a KV store that uses LSMT data structures
- The data structure for the leaf nodes is contained in 8 files
  - Data.db is the largest
  - In this example the sst size was set to 256MB
- Other LSMT implementation use a single file for each leaf node

```
 32K Aug 26 21:57 md-4127-big-CompressionInfo.db
257M Aug 26 21:57 md-4127-big-Data.db
  10 Aug 26 21:57 md-4127-big-Digest.crc32
 20K Aug 26 21:57 md-4127-big-Filter.db
454K Aug 26 21:57 md-4127-big-Index.db
 11K Aug 26 21:57 md-4127-big-Statistics.db
4.6K Aug 26 21:57 md-4127-big-Summary.db
  92 Aug 26 21:57 md-4127-big-TOC.txt
 32K Aug 26 21:57 md-4152-big-CompressionInfo.db
257M Aug 26 21:57 md-4152-big-Data.db
  10 Aug 26 21:57 md-4152-big-Digest.crc32
 22K Aug 26 21:57 md-4152-big-Filter.db
453K Aug 26 21:57 md-4152-big-Index.db
 11K Aug 26 21:57 md-4152-big-Statistics.db
4.6K Aug 26 21:57 md-4152-big-Summary.db
  92 Aug 26 21:57 md-4152-big-TOC.txt
 32K Aug 26 21:59 md-4340-big-CompressionInfo.db
257M Aug 26 21:59 md-4340-big-Data.db
  10 Aug 26 21:59 md-4340-big-Digest.crc32
 20K Aug 26 21:59 md-4340-big-Filter.db
453K Aug 26 21:59 md-4340-big-Index.db
 11K Aug 26 21:59 md-4340-big-Statistics.db
4.6K Aug 26 21:59 md-4340-big-Summary.db
  92 Aug 26 21:59 md-4340-big-TOC.txt
```

Test Workload

- YCSB test
  - Thread count = 1
- Run Phase done with Uniform Distribution
  - Note: Default setting is for Zipfian Distribution
- Workload
  - 15 Million Key loads followed by 60 Million random puts
  - Compaction threads set to 8
  - SST file size = 160MB

Data Collection on Data.db files

- A modified version Linux's inotifywait was used to track the lifecycle of the files
  - The modified version increased the time resolution to from a second to ns
- Timestamps were collected for open, close and delete events for each Data.db file

```
inotifywait  --format '%T %f %e' --timefmt %s -r  -m -e  create -e close_write -e delete -e attrib /a/cassandra/data/ycsb
```

Man Page

Name

inotifywait - wait for changes to files using inotify

Synopsis

inotifywait [**-hcmrq**] [**-e** <event>] [**-t** <second

Description

inotifywait efficiently waits for changes to files using Linux's ***inotify(7)*** interface. It is suitable for waiting for changes to files from shell scripts. It can either exit once an event occurs, or continually execute and output events as they occur.

Output

inotifywait will output diagnostic information on standard error and event information on standard output. The event output can be configured, but by default it consists of lines of the following form:

watched_filename EVENT_NAMES event_file

Events

The following events are valid for use with the **-e** option:

access
A watched file or a file within a watched directory was read from.

modify
A watched file or a file within a watched directory was written to.

attrib
The metadata of a watched file or a file within a watched directory was modified. This includes timestamps, file permissions, extended attributes etc.

close_write
A watched file or a file within a watched directory was closed, after being opened in writeable mode. This does not necessarily imply the file was written to.

close_nowrite
A watched file or a file within a watched directory was closed, after being opened in read-only mode.

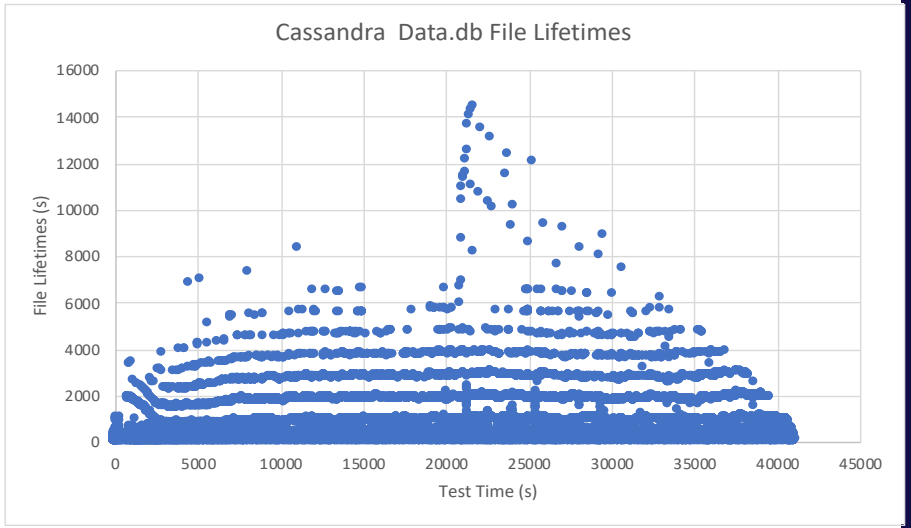
close
A watched file or a file within a watched directory was closed, regardless of how it was opened. Note that this is actually implemented simply by listening for both **close_write** and **close_nowrite,** hence all close events received will be output as one of these, not **CLOSE.**

open
A watched file or a file within a watched directory was opened.

moved_to
A file or directory was moved into a watched directory. This event occurs even if the file is simply moved from and to the same directory.

Data.db File Lifetimes

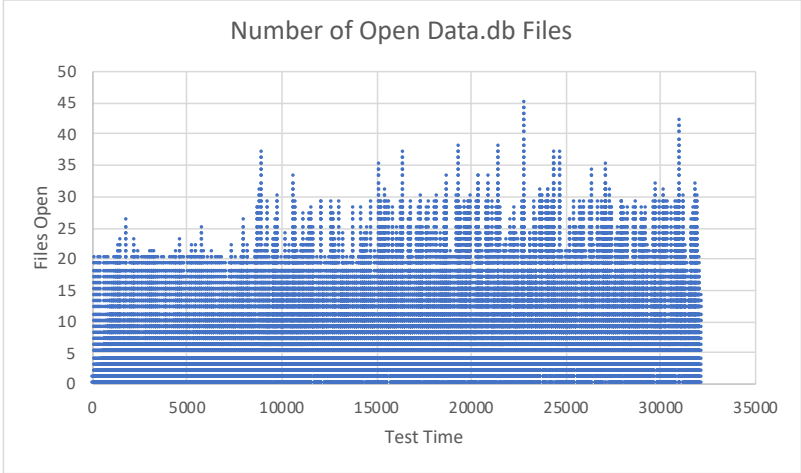
- Overall the File Lifetimes look as expected
- Each level of the Cassandra Merge tree is expected to have a longer lifetime than the previous level
  - The highest level should have the longest life
  - The Lowest level should change the most
- The use of lifetimes of SST files for data placement and the reduction in WAF was shown by Taejin Kim in Fast 19



https://www.usenix.org/system/files/fast19-kim-taejin.pdf

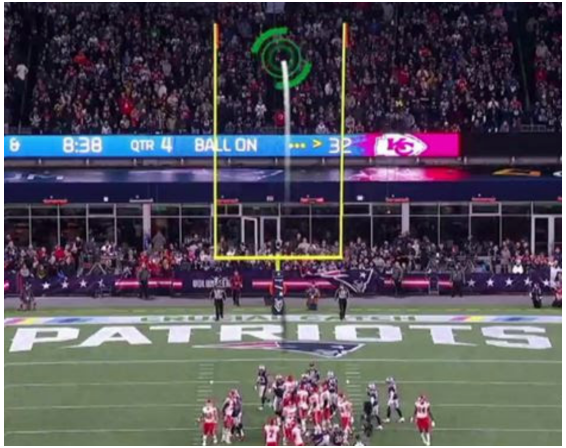
Number of Open Files

- The total number of open files gives an indication how segments zones are required to support Data Placement by SST file(s)
  - If Cassandra SST files were compacted into a single file, ~ 45 zones would be required to support Data Placement by SST file
    - Higher put workload increases the number of Open Files



Life is not Perfect

- File sizes will vary, there will be no perfect alignment between files and storage blocks
- There are two interesting options that define the goal posts
  - Left goal post
    - Maintain spatial locality of immutable files
  - Right goal post
    - Maintain spatial locality of immutable files
    - Utilize file characteristics to assist in data placement of files



https://www.sportsvideo.org/2018/10/25/nbc-sports-to-debut-new-field-goal-tracer-graphic-on-sunday-night-football/

Left Goal Post

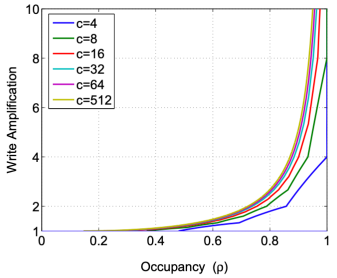


Figure 8: Write amplification A as a function of ρ for $c = 4, 8, 16, 32, 64,$ and 512.

Performance of the Greedy Garbage-Collection Scheme in Flash-Based Solid-State Drives

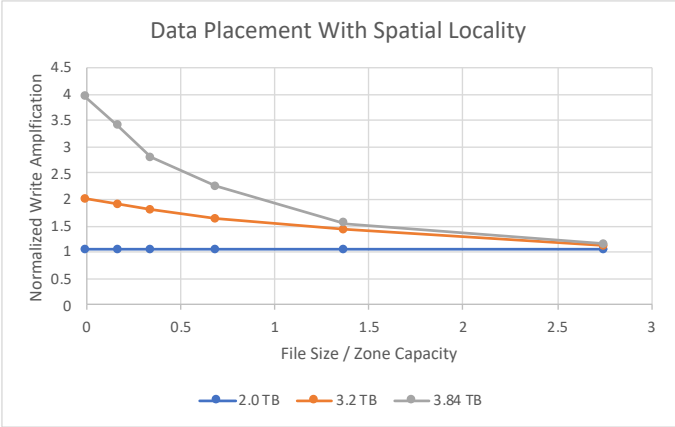
Ilias Iliadis

IBM Research – Zurich, 8803 Rüschlikon, Switzerland
Phone: +41-1-724-8646; Fax: +41-1-724-8952; e-mail: ili@zurich.ibm.com

https://dominoweb.draco.res.ibm.com/reports/rz3769.pdf

Spatial Locality of Large Block Write

- Without spatial locality maintained by an SSD immutable file's writes amplification are the point along the y- axis
- If a WA = <2 is required to scale performance and the immutable file sizes are ~equal to the Zone Capacity
- Without Spatial locality
  - 3.20 TB user capacity
- With ZNS Spatial locality
  - 3.84TB of user capacity



Imaginary Generic System

Any Application with Large Immutable Files

LSF

Maintains Spatial locality of data files

ZNS Storage Device

Maintains Spatial locality of data files

Right Goal Post

- Goal: end-to-end ZNS integration
  - Add native support for ZNS as a RocksDB FileSystem class
  - Use as much as possible of RocksDB data knowledge to do smart data placement
- Minimize write amplification
  - Avoid garbage collection
  - Minimize wear
  - Maximize write throughput
  - Improve read performance
- Minimize integration effort for users

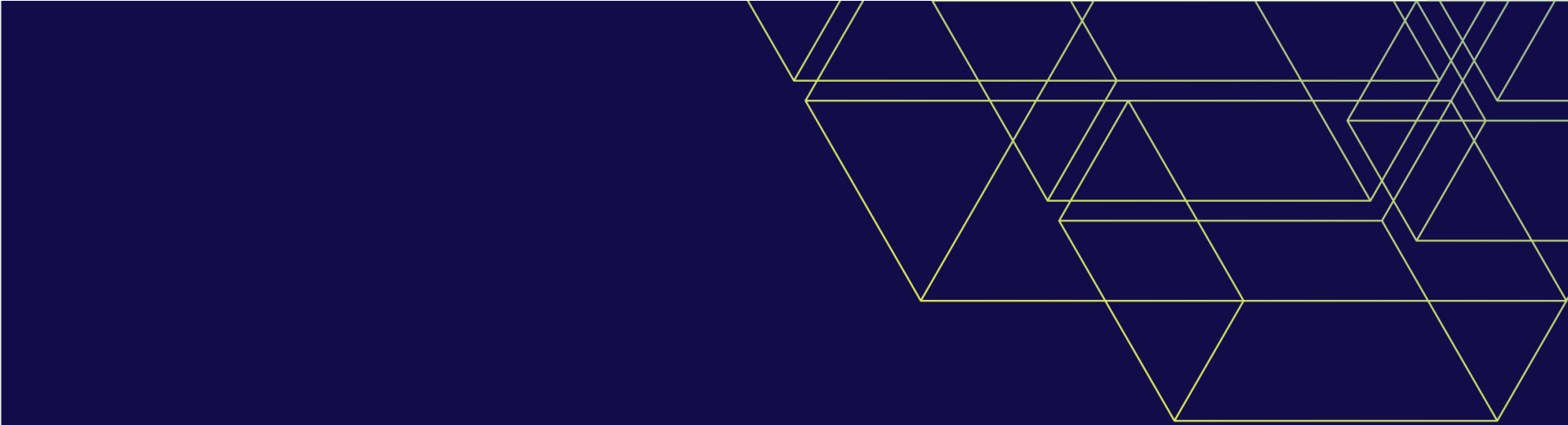
ZenFS, Zones and RocksDB
Who likes to take out the garbage anyway?

Hans Holmberg

Summary

- File systems and applications that use large immutable files can see increases in performance and user capacity
  - Large immutable files are efficient for both applications and storage media
- Traditional SSDs using LBA addressing do not maintain spatial locality on the media for files
  - Traditional SSDs require OP to scale performance in large block immutable files
- ZNS maintains spatial localities and restore the efficiency of using large immutable files

marc.acosta@wdc.com



Please take a moment to rate this session.

Your feedback matters to us.