IO Pattern based Optimization in SSD

LU Xiangfeng

CTO, Memblaze Co., Ltd.

IO Pattern based Optimization

- IO patterns are important to SSD optimization
  - To determine best location to store data, optimize unnecessary garbage collection, reduce write amplification.
  - For fast data access, reduce the latency in future reads.
- Enterprise applications can benefit from SSD IO pattern based optimizations.
  - Open source society actively pursuits application level hints as guidance to improve SSD performance and endurance, while lowering TCO.

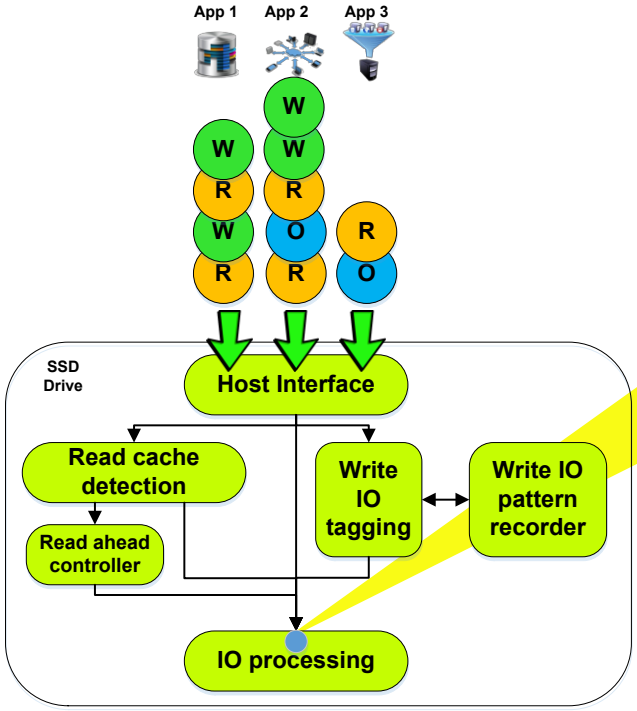
Optimization challenges for SSD

- Application requirement challenges
  - SDS and cloud applications use SSD as pool. IOs from different applications are often mixed together and directed to SSD. SSD capacity continues to grow, which accelerates this trend.
  - Today's applications often pose strict requirement for SSD access latency. Optimization should not sacrifice IO latency and service quality.
- SSD design constraints challenges
  - Mapping IOs to separated flash blocks based on pattern becomes expensive because flash geometry changes in 3D TLC.
  - IO Optimization should intelligently adapt to different workloads to balance between short term target and long term target.

IO Pattern Optimization Basic Idea

- IO pattern tagging, detection and prediction
  - Provide necessary hints for SSD to arrange each IO properly. Hints can be passed down from host or learnt from history records.
- Resource allocation coordination service
  - Because of resource limitation for flash open page, RAID buffer, overlapped programming IOs, a centralized coordination service is set up to manage all resources.
- Intelligent scenario detection and adaption
  - Resource allocation coordination service lacks global knowledge, so a scenario detection unit responsible for compiling high level information, and instructs the coordination service to change certain strategies to adapt to new scenario.

Frontend IO Pattern Recognition



	Action
Read	Read and cache
	Read without cache
	Read ahead and cache
Write	Write tag 0
	Write tag 1
	…
	Write tag N-1

IO commands are sent to frontend detection and prediction unit.

- Read requests, based on historical statistics of this application type, are tagged either need caching or not.
- Write requests are tagged with different IDs based on LBA affinity and IO size.
- Other requests are passed down untouched.

IO Requests Annotation

From standard definitions for IO request hints - T10, T13, POSIX, Linux

M.3 Access Patterns LBMDs

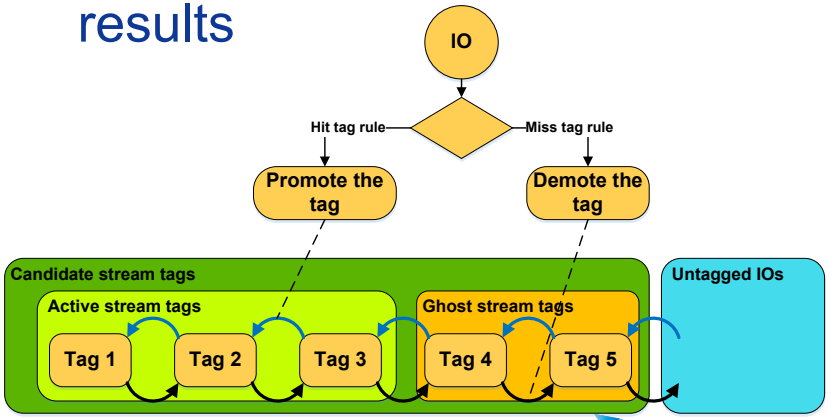
M.3.1 Access Patterns LBMD format for SCSI

The Access Patterns LBMD format processed by SCSI device servers is shown in table M.4.

Table M.4 — Access Patterns LBMD format for SCSI

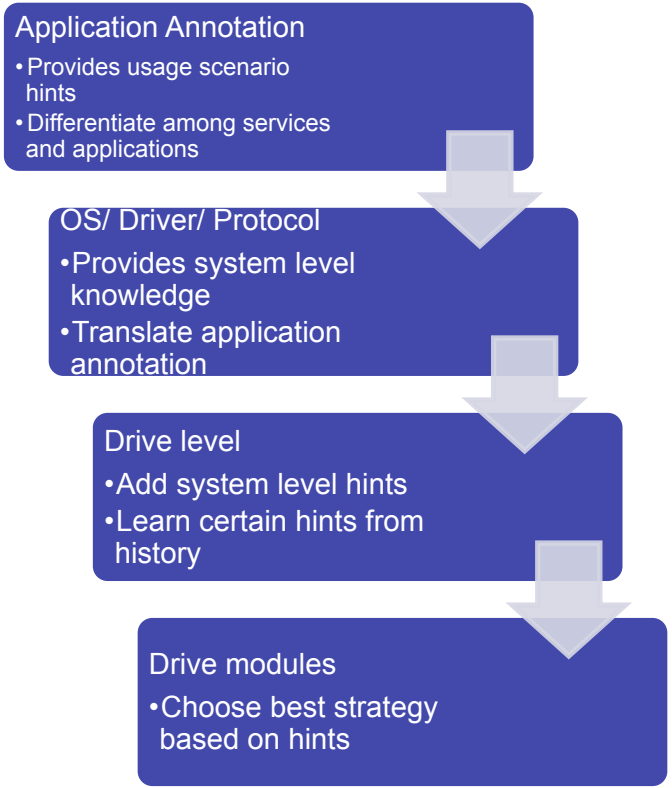
Bit / Byte	7	6	5	4	3	2	1	0
0	ACDLU	Reserved			LBMD TYPE (0h)			
1	OVERALL FREQUENCY		READ/WRITE FREQUENCY		WRITE SEQUENTIALITY		READ SEQUENTIALITY	
2	Reserved				SUBSEQUENT I/O		OSI PROXIMITY	
3	Reserved							

Dynamically learnt from IO attributes or historical statistics - Rely on rule sets and machine learning results



IO Request Annotation Types

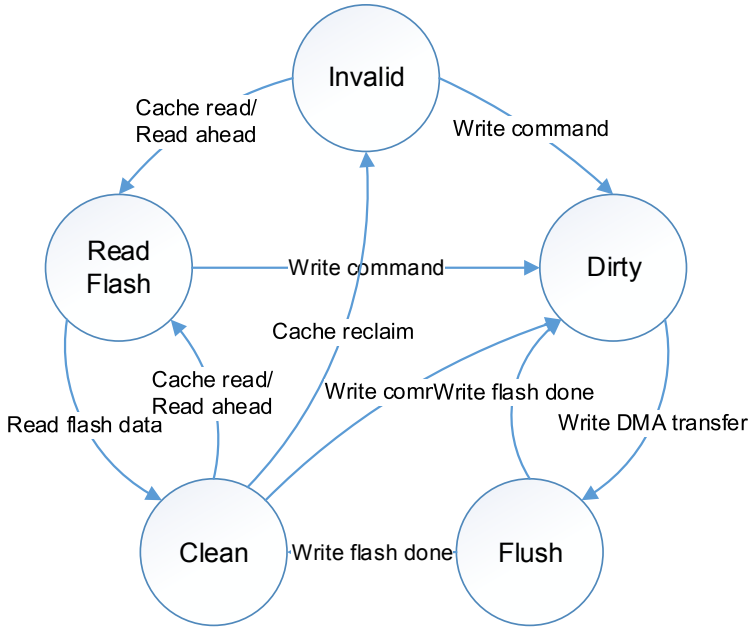
Annotation Types	Description
Read Frequency	If SSD internally use RAM to implements cache. With read frequency annotation, firmware can decide if the data is hot data and place hot data in cache.
Read Sequentiality	For sequential read IOs, drive can choose to trigger read ahead and optimize latency. It can also trigger read cache inside flash. For random read IOs, drive only fetch a portion of flash page data, save the time cost during flash read.
Write Frequency	Drive internally may use multiple storage tiers. With write frequency annotation, firmware can choose the optimal storage tier. Besides, it can arrange hot data and cold data to different physical locations to improve GC efficiency.
Write Sequentiality and Tagging	Sequential write data should be stored in a page or neighbor pages. This can benefit GC. Tagging is used to differentiate among IOs from different applications, or different services. Data with different tagging should be physically separated.
IO Priority and Latency Hints	IO priority and latency hints are used to instruct drive to prioritize IO service quality. The drive could over-provision certain bandwidth to favor real time and high priority IOs.



Volatile Read Cache

Inside drive, a volatile read cache can help IOs with high frequency hints. Drive can also trigger read ahead request if necessary.

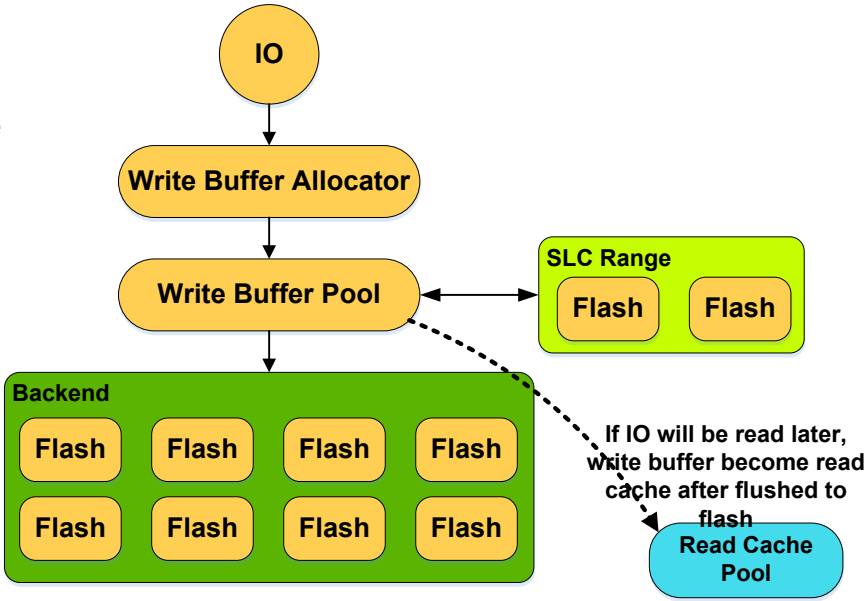
- Read cache line size is set to 4kB to balance the line granularity and storage size.
- Each cache line can be in any of 5 states.
  - Invalid – Initial state.
  - Read Flash – Drive issue read request to backend, data is not available yet.
  - Clean – Data is available in read cache.
  - Dirty – A write request hit the cache line, data is not available yet.
  - Flushing – A write request hit the cache line and data is stored in cache memory, data acknowledged to host but not flushed to backend yet.
- Read cache can be invalidated at any time without loss of data. (Acknowledged data is specially handled in non-volatile write buffer unit) ARC is used as cache retirement policy.



Non Volatile Write Buffer

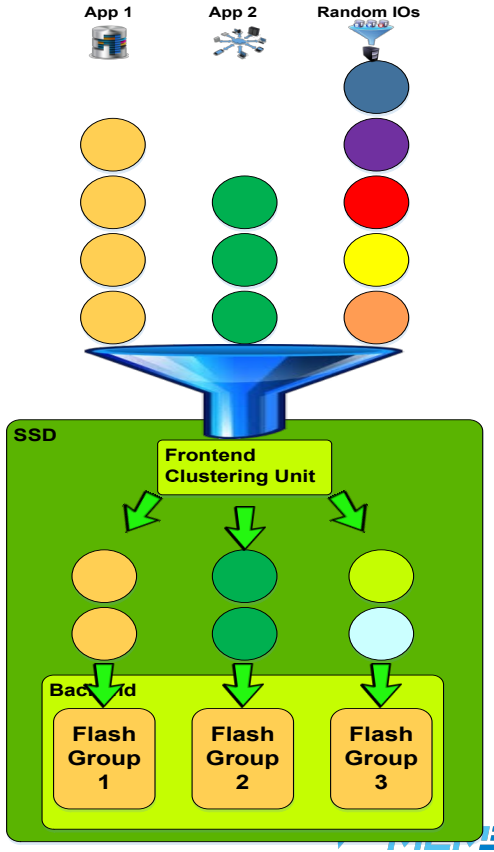
SSD can implement non volatile write buffer. For write IO requests, data can be stored in this buffer so that acknowledgement to host can preempt data written to flush. Non volatile write buffer usually uses DDR to store data at normal condition, and quickly flushes data to SLC tier in SSD upon power loss.

- IOs with hints indicating it may be written again in near future can hold off writing data to flash.
- With hint indicating high probability read in near future, data can reside in DDR without invalidation.
- Sequential write requests are aggregated before sending to backend. Backend can then arrange them in neighbor places, improving backend flash write efficiency and facilitating GC.



Write IO Tagging & Clustering

- Separating write IOs from different applications or services
  - Tag passed down from host applications or OS.
  - IO tag from clustering result.
- Place IOs with same tag together in a flash block or recycle unit
  - IOs with same tag are likely to be retired at same time. So save GC's efforts to move data.
  - Require a lot of open blocks that can accept write requests simultaneously.
  - For pure random write IOs, assign a special tag to them.



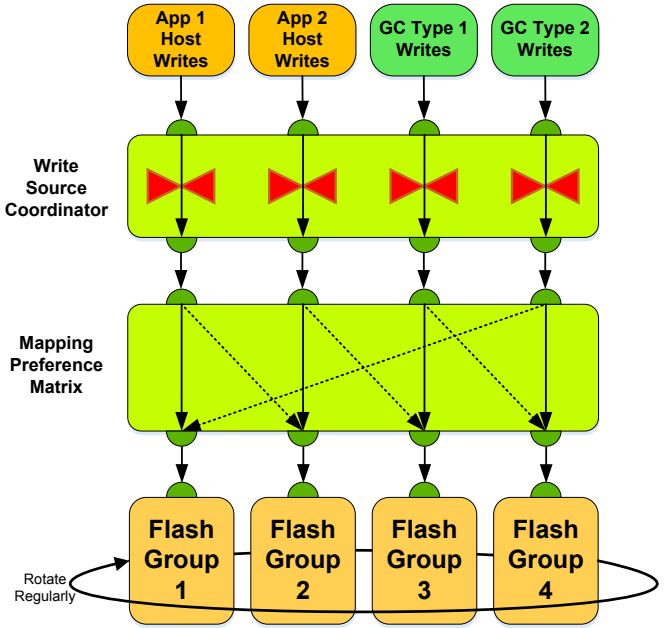
Auto Adjustment of Bandwidth

- Only limited numbers of flash blocks can be opened at a time due to physical limitation
  - Today's nand flash is prone to error if the block is not fully programmed. SSD need to cache quite a lot of data in DDR.
  - SSD need to flush all data acknowledged but not fully written to flash. So the dirty data size should be small enough.
- Adjust the flash group mapping relation with tag to optimize flash bandwidth
  - With limited open block resource, the mapping between open flash block to tag should not be static. When the write intensity for a tag is changed, its occupancy of flash block should also change. This policy can maximize the flash channel bandwidth.

Auto Adjustment of Bandwidth (Cont'd)

Assign tagged IO requests to flash lun groups

- A frontend unit controls the bandwidth for write requests in each tag and the GC bandwidth. This policy coordinates each user application writes and GC writes intensity.
- Partition flashes into flash lun groups. Each lun group serves one tag, if it has additional bandwidth, it can serve other tag types. This policy keeps number of open flash blocks low and maintain all open blocks busy if necessary.
- Regularly rotate the map between tag and flash lun group. This policy ensures the data in each tag category is evenly distributed among all flash lun group. GC can then transfer data efficiently.



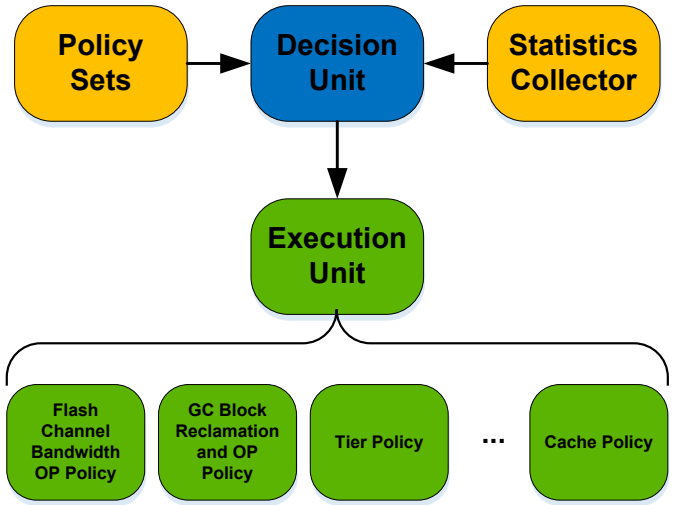
Optimize Flash Access Behaviors

- Use of latency and priority hints
  - For host read with high priority or low latency, can place the request in high priority queue so that it can be served quickly.
  - Use program suspension to temporarily suspend an in progress program command execution inside flash packet and let a read request precedes.
- Use of sequentiality hints
  - For sequential read requests, if they are stored in physically contiguous address, then use (multiplane) page read to read out all data. For random read requests, only instruct flash read the necessary part inside a page.
  - Data can be reside in nand flash cache registers for later usage.

Scenario detection and adaption

Use scenario detection and feedback to adjust SSD's configuration lively

- User can set their expectation on SSD, SSD can also dynamically learn user behavior via a statistics collection unit.
- Decision unit can aggregate these knowledge and configure the execution unit to adapt to user's current scenario.
  - Set certain parameters in each execution unit. (i.e. Adjust cache policy paramters)
  - Configure the system from a bunch of optional policy. (i.e. If user prefer low latency rather than high throughput, or SSD learns it via hints, then SSD can adjust the backend queue policies to reduce latency.)



Conclusion

- IO pattern tagging, detection and prediction can benefit SSD's design and better meet user's requirement.
- Through structural design of SSD, the IO pattern hints can guide SSD optimization. Structural design coordinates different modules inside SSD to maximize their utility.
- Scenario detection and adaption can help to tune the overall configuration to adjust SSD working behavior best fits user scenario.

Promotion

- Please visit our booth at #319.
- LU Xiangfeng
- CTO, Memblaze Co., Ltd.