Preview

- Solid State Driver
  - Structure of Solid State Driver
  - Architecture of a SSD
  - Basic Operations in SSD (read, write erase)
  - Flash Translation Layer (FTL)
  - Garbage Collection

Structure of an SSD

- Solid-state drives (SSD) is a flash-memory based data storage device. Bits are stored into cells, which are made of **floating-gate transistors**. (HDD: magnetic storage)
- SSDs are made entirely of electronic components, there are no mechanical parts like in hard disk drivers.
- Voltages are applied to the floating-gate transistors, which is how bits are being read, written, and erased. Two solutions exist for wiring transistors: the NOR flash memory, and the NAND flash memory.
- An important property of NAND-flash modules is that their cells are wearing off, and therefore have a limited lifespan.

Structure of an SSD

- NAND Flash Memory
  - Smaller Cell size
  - Slow to read
  - Faster to erase and write
  - Less expensive
  - Higher memory capacity
  - Has lifespan
  - Used in devices to which large files are frequently uploaded and replaced.
- NOR Flash Memory
  - Larger Cell size
  - Faster to read
  - Slower to erase and write
  - Much more expensive
  - Need more power consumption when power on
  - Used in mobile phones, scientific instruments and medical devices.

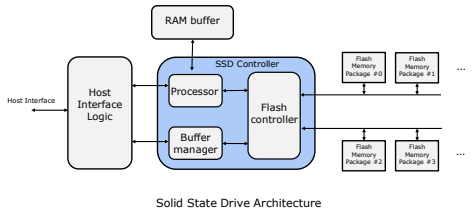
Structure of an SSD

- Each cell has a maximum number of P/E cycles (Program/Erase).
- NAND-flash memory wears off and has a limited lifespan. The lifespan of SSDs could be tremendously increased.
- Types of cells
  - Single level Cell - transistors can store only 1 bit but have a long lifespan
  - Multiple level Cell -transistors can store 2 bits, at the cost of a higher latency and reduced lifespan compared to SLC
  - Triple level Cell –transistors can store 3 bits, but at an even higher latency and reduced lifespan.
- Having more bits for the same amount of transistors reduces the manufacturing costs. SLC-based SSDs are known to be more reliable and have a longer life expectancy than MLC-based SSDs, but at a higher manufacturing cost.

Structure of an SSD
(NAND- Flash pages and Blocks)

- Cells are grouped into a grid, called a **block**, and blocks are grouped into **planes**.
- The smallest unit through which a block can be read or written is a **page**.
- Pages cannot be erased individually, only whole blocks can be erased.
- The size of a NAND-flash page size can vary, and most drive have pages of size 2 KB, 4 KB, 8 KB or 16 KB.
- Most SSDs have blocks of 128 or 256 pages, which means that the size of a block can vary between 256 KB and 4 MB.

Architecture of a SSD



Solid State Drive Architecture

Architecture of a SSD

- Commands (read/write) come from the user through the host interface.
- The processor in the SSD controller takes the commands and pass them to the flash controller.
- SSDs also have embedded RAM memory, generally for caching purposes and to store mapping information (mapping tables).
- The packages of NAND flash memory are organized in gangs, over multiple channels.

Basic Operations in SSD (read, write erase)

- Due to the organization of NAND-flash cells, it is not possible to read or write single cells individually.
- Memory is grouped and is accessed with very specific properties.
- Reads are aligned on page size
  - It is not possible to read less than one page at once.
  - One can request just one byte from the operating system, but a full page will be retrieved in the SSD, forcing a lot more data to be read than necessary(read amplification).

Basic Operations in SSD (read, write erase)

- Writes are aligned on page size
  - When writing to an SSD, writes happen by increments of the page size. So even if a write operation affects only one byte, a whole page will be written anyway.
  - Writing more data than necessary is known as write amplification.
- Pages cannot be overwritten
  - A NAND-flash page can be written to only if it is in the "free" state.
  - When data is changed, the content of the page is copied into an internal register, the data is updated, and the new version is stored in a "free" page, an operation called "read-modify-write".

Basic Operations in SSD (read, write erase)

- Erases are aligned on block size
  - Pages cannot be overwritten, and once they become stale, the only way to make them free again is to erase them.
  - It is not possible to erase individual pages, and it is only possible to erase whole blocks at once – one page modification in a block need the entire block moving to a free block
  - The erase command is triggered automatically by the garbage collection process in the SSD controller when it needs to reclaim stale pages to make free space.

Basic Operations in SSD (read, write erase)

Block #1000 with data

PPN	data
0	x
1	y
2	z
3	

Block #2000 : free

PPN	data
0	
1	
2	
3	

Block #2000 is free
Block #1000 is with data saved in PPN(physical page number) 0, 1, 2 and one free page at PPN =3 in Block #1000

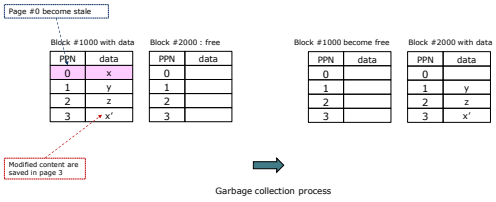
A page size could be 2KB, 4KB, 8KB or 16KB

Lets assume data in page 0 in Block #1000 is modified. Since page cannot be overwritten, updated data must write to free page in the block. Then garbage collection process do the followings.

1. The valid pages from the data block 1000 into the free block (2000).
2. Block 1000 is erased become free block

Blocks can only be erased a limited number of times based on P/E cycle until they wear off.

Basic Operations in SSD (read, write erase)



Basic Operations in SSD (read, write erase)
Write Amplification

- Because writes are aligned on the page size, any write operation that is not both aligned on the page size and a multiple of the page size will require more data to be written than necessary, a concept called write amplification.
- Writing one byte will end up writing a page, which can amount up to 16 KB for some models of SSD and be extremely inefficient.
- Writing data in an unaligned way causes the pages to be read into cache before being modified and written back to the drive, which is slower than directly writing pages to the disk. This operations is known as read-modify-write

Basic Operations in SSD (read, write erase)
(Write Amplification)

- Never write less than a page
  - Avoid writing chunks of data that are below the size of a NAND-flash page to minimize write amplification and prevent read-modify-write operations.
  - To maximize throughput, whenever possible, keep small writes into a buffer in RAM and when the buffer is full, perform a single large write to batch all the small writes.

Basic Operations in SSD (read, write erase)
(Wear Leveling)

- NAND-flash cells have a limited lifespan due to their limited number of P/E cycles.
- What if data was always read and written from the same exact block. This block would quickly exceed its P/E cycle limit, wear off, and the SSD controller would mark it as being unusable.
- One of the main goals of an SSD controller is to implement wear leveling by careful block selection for writing, which distributes P/E cycle as evenly possible among the locks by selecting blocks carefully when writing.
- Ideally, all blocks would reach their P/E cycle limits and wear off at the same time.

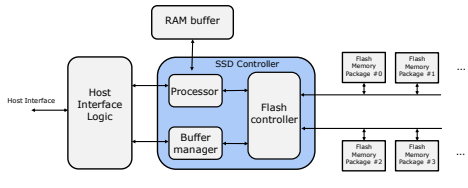
Flash Translation Layer (FTL)

- The main factor that made adoption of SSDs so easy is that they use the same host interfaces as HDDs.
- Although presenting an array of Logical Block Addresses (LBA) makes sense for HDDs as their sectors can be overwritten, it is not fully suited to the way flash memory works.
- For this reason, an additional component is required to hide the inner characteristics of NAND flash memory and expose only an array of LBAs to the host.
- This component is called the Flash Translation Layer (FTL), and resides in the SSD controller. The FTL is critical and has two main purposes: logical block mapping and garbage collection.

Flash Translation Layer (FTL)
(Logical Block Mapping)

- The logical block mapping translates logical block addresses (LBAs) from the host space into physical block addresses (PBAs) in the physical NAND-flash memory space.
- This mapping takes the form of a table, which for any LBA gives the corresponding PBA.
- This mapping table is stored in the RAM of the SSD for speed of access, and is persisted in flash memory in case of power failure. When the SSD powers up, the table is read from the persisted version and reconstructed into the RAM of the SSD.

Flash Translation Layer (FTL)
(Logical Block Mapping)



Solid State Drive Architecture

Flash Translation Layer (FTL)
(Logical Block Mapping)

- The naive approach is to use a **page-level mapping** to map any logical page from the host to a physical page.
- This mapping policy offers a lot of flexibility, but the mapping table could be extremely large, which increase the manufacturing costs (need larger RAM).
- A solution to that would be to map blocks instead of pages, using a **block-level mapping**.
- An SSD drive has 256 pages per block with block-level mapping requires 256 times less memory than page-level mapping, which is a huge improvement for space utilization.
- However, the mapping still needs to be persisted on disk in case of power failure, and in case of workloads with a lot of small updates, full blocks of flash memory will be written whereas pages would have been enough. This increases the write amplification and makes block-level mapping widely inefficient

Flash Translation Layer (FTL)
(Logical Block Mapping: Log Block Mapping)

- The tradeoff between page-level mapping and block-level mapping is the one of performance versus space.
- The log block mapping use hybrid approach between page-level and block-level mapping which uses an approach similar to log-structured file system.
- Incoming write operations are written sequentially to log blocks. When a log block is full, it is merged with the data block associated to the same logical block number (LBN) into a free block.
- Only a few log blocks need to be maintained, which allows to maintain them with a page granularity

Flash Translation Layer (FTL)
(Logical Block Mapping)

- Figure shows a simplified representation of a hybrid log-block FTL, in which each block only has four pages.
- Four write operations are handled by the FTL, all having the size of a full page.
- The logical page numbers (LPN) of 5 and 9 both resolve to LBN=1, which is associated to the physical block #3000 (see Data Block Mapping Table).
- Initially, all the physical page offsets are null at the entry where LBN=1 is in the log-block mapping table, and the log block #1000 is entirely empty as well.
- The first write, b' at LPN=5, is resolving to LBN=1 by the log-block mapping table, which is associated to PBN=1000 (log block #1000).
- The page b' is therefore written at the physical offset 0 in block #1000.
- The metadata for the mapping now needs to be updated, and for this, the physical offset associated to the logical offset of 1 (arbitrary value for this example) is updated from null to 0.

Flash Translation Layer (FTL)
(Logical Block Mapping)

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	
		2	
		3	
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000
2	5000
3	4000
...	...
7	8000

Free block mapping table

PBN
9000
7000
...

Block 1000(log)

PPN	data
0	
1	
2	
3	

Block 3000(data)

PPN	data
0	a
1	b
2	c
3	d

Block 9000(free)

PPN	data
0	
1	
2	
3	

Flash Translation Layer (FTL)
(Logical Block Mapping)

1. Write (5, b')

Sequence of four write operations.
The logical addresses 5 and 9 are both translated into the LBN =1

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	0
		2	
		3	
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000
2	5000
3	4000
...	...
7	8000

Free block mapping table

PBN
9000
7000
...

Block 1000(log)

PPN	data
0	b'
1	
2	
3	

Block 3000(data)

PPN	data
0	a
1	b
2	c
3	d

Block 9000(free)

PPN	data
0	
1	
2	
3	

Flash Translation Layer (FTL)
(Logical Block Mapping)

1. Write (5, b')
2. Write (9, d')

Sequence of four write operations.
The logical addresses 5 and 9 are both translated into the LBN =1

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	0
		2	
		3	1
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000
2	5000
3	4000
...	...
7	8000

Free block mapping table

PBN
9000
7000
...

Block 1000(log)

PPN	data
0	b'
1	d'
2	
3	

Block 3000(data)

PPN	data
0	a
1	b
2	c
3	d

Block 9000(free)

PPN	data
0	
1	
2	
3	

Flash Translation Layer (FTL)

(Logical Block Mapping)

1. Write (5, b')
2. Write (9, d')
3. Write (9, d")

Sequence of four write operations. The logical addresses 5 and 9 are both translated into the LBN =1

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	0
		2	
		3	1 ->2
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000
2	5000
3	4000
...	...
7	8000

Become stale

Block 1000(log)

PPN	data
0	b'
1	d'
2	d"
3	

Block 3000(data)

PPN	data
0	a
1	b
2	c
3	d

Free block mapping table

PBN
9000
7000
...

Block 9000(free)

PPN	data
0	
1	
2	
3	

Flash Translation Layer (FTL)

(Logical Block Mapping)

1. Write (5, b')
2. Write (9, d')
3. Write (9, d")
4. Write (5, b")

Sequence of four write operations. The logical addresses 5 and 9 are both translated into the LBN =1

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	0 ->3
		2	
		3	1 ->2
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000
2	5000
3	4000
...	...
7	8000

Become stale

Block 1000(log)

PPN	data
0	b'
1	d'
2	d"
3	b"

Block 3000(data)

PPN	data
0	a
1	b
2	c
3	d

Now log is full, need merge into free block

Free block mapping table

PBN
9000
7000
...

Block 9000(free)

PPN	data
0	
1	
2	
3	

Flash Translation Layer (FTL)

(Logical Block Mapping)

1. Write (5, b')
2. Write (9, d')
3. Write (9, d")
4. Write (5, b")

Sequence of four write operations. The logical addresses 5 and 9 are both translated into the LBN =1

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	0 ->3
		2	
		3	1 ->2
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000->9000
2	5000
3	4000
...	...
7	8000

Become stale

Block 1000(log)

PPN	data
0	b'
1	d'
2	d"
3	b"

Block 3000(data)

PPN	data
0	a
1	b
2	c
3	d

Merge log block and a data block into a free block

Free block mapping table

PBN
9000
7000
...

Block 9000(free)

PPN	data
0	a
1	b"
2	c
3	d"

After merging, Block 1000 and Block 3000 become free block

Flash Translation Layer (FTL)

(Logical Block Mapping)

1. Write (5, b')
2. Write (9, d')
3. Write (9, d")
4. Write (5, b")

Sequence of four write operations. The logical addresses 5 and 9 are both translated into the LBN =1

Log-block page mapping table

LBN	PBN	Logical page offset	Physical page offset
1	1000	0	
		1	
		2	
		3	
...	...	...	...
7	2000	...	...

Data block mapping table

LBN	PBN
0	6000
1	3000->9000
2	5000
3	4000
...	...
7	8000

Block 1000(free)

PPN	data
0	
1	
2	
3	

Block 3000(free)

PPN	data
0	
1	
2	
3	

Now Block 1000 and 3000 become free by garbage collection

Free block mapping table

PBN
3000
7000
...

Block 9000(data)

PPN	data
0	a
1	b"
2	c
3	d"

After merging, Block 1000 and Block 3000 become free block

Flash Translation Layer (FTL)

(Logical Block Mapping)

- The write operations go on and the mapping metadata is updated accordingly. When the log block #1000 is entirely filled, it is merged with the data block associated to the same logical block, which is block #3000 in this case.
- This information can be retrieved from the data-block mapping table, which maps logical block numbers to physical block numbers.
- The data resulting from the merge operation is written to a free block, #9000 in this case. When this is done, both blocks #1000 and #3000 can be erased and become free blocks, and block #9000 becomes a data block.
- The metadata for LBN=1 in the data-block mapping table is then updated from the initial data block #3000 to the new data block #9000.

Garbage Collection

- Since pages cannot be overwritten, if the data in a page has to be updated, the new version is written to a *free* page, and the page containing the previous version is marked as *stale*.
- When blocks contain stale pages, they need to be erased before they can be written to.
- Because of the high latency required by the erase command compared to the write command, this extra erase step incurs a delay which makes the writes slower.
- Some controllers implement a *background garbage collection process*, which takes advantage of idle time and runs regularly in the background to reclaim stale pages and ensure that future foreground.