Smart SSD: Host와 Storage의 수직 최적화 Solution을 위한 신개념 SSD

HDD (Hard Disk Drive)와 동일한 기능을 제공하면서 빠른 성능과 높은 신뢰성을 제공하는 SSD는 기술 성숙기에 들어섰다. 빠른 SSD를 고려한 storage 솔루션들이 이미 시장에 널리 퍼지고 있다. 하지만, SSD를 단순히 빠른 저장장치로만 쓰기에는 SSD가 가지고 있는 리소스가 아깝다. SSD는 이미 내부에 여러 개의 CPU와 큰 크기의 DRAM을 가지고 있고, 전용 HW accelerator를 추가로 탑재할 수 있으며, 이를 통해 HDD가 제공하기 어려운 새로운 기능들을 지원할 수 있다. 이제는 SSD에 새로운 기능을 추가하고, Host와 Storage의 수직 최적화를 통해 더욱 효과적으로 또는 Smart 하게 SSD를 사용하는 방법을 고민해야 할 때이다. SSD에 전통적인 read/write 이외의 새로운 기능을 추가함으로써 부가가치를 높이려는 많은 시도가 있다. SSD가 제공하는 새로운 기능을 활용하려면 host SW 또한 변경되어야 한다. 본 논문에서는 HDD의 대체 저장장치에서 출발한 SSD가 어떤 방향으로 진화하고 있는지 설명한다.

Ⅰ. 서 론

HDD (Hard Disk Drive)가 빠르게 SSD (Solid State Drive)로 대체되고 있다. SSD는 random write와 random read의 성능이 매우 우수하며 latency 또한 매우 짧아서 사용자가 느끼는 컴퓨터의 응답성을 크게 개선시킬 수 있다. 빠른 응답은 사용자의 컴퓨터 사용 만족도를 높이기 때문에 SSD의 인기가 날로 높아지고 있다.

기존의 SSD가 "빠른 HDD"를 목표로 발전해 왔다면, 최근의 환경 변화는 SSD를 다른 용도로 사용하려는 시도에 대한 당위성을 높이고 있다. Storage interface는 기존의 SATA III 6Gbps(약 540MB/s)에서 PCIe Gen III 4 lane의 약 3200MB/s까지 유효

권 문 상
삼성전자 메모리 사업부
Solution 개발실

bandwidth가 증가하였고, interface latency도 수 us로 줄었으며, SSD 내의 controller가 사용하는 CPU의 개수도 증가하고 clock 속도도 증가하고 있다. 또한, power loss protection 기능을 포함하는 SSD는 non-volatile 하게 write 해야 할 data를 RAM에 어느 정도 보존할 수 있어서 synchronous write에 대해서도 순간적으로 빠르게 처리할 수 있는 능력을 갖추게 되었다. SSD가 computer architecture 상의 PCIe bus에 연결됨으로써 bus master가 되어 system DRAM을 접근할 수 있게 된 것도 큰 변화라고 할 수 있다.

본 논문에서 기술하는 'Smart SSD'란 2가지 의미를 가진다. 하나는 기존의 SSD가 제공하지 않는 새로운 기능을 제공함으로써 성능과 수명, 비용 관점에서 좀 더 효율적으로 동작하는 SSD라는 의미고, 다른 하나는 SSD가 제공하는 새로운 특성들을 효율적으로 이용하는 host SW system을 의미한다. 본 논문에서는 SSD의 새로운 발전 방향들을 살펴본다. 본 논문은 다음과 같이 구성되어 있다. 본론에서는 SSD가 HDD를 대체하는 더 효율적인 저장장치로써 발전하기 위한 노력, SSD를 DRAM의 확장으로 사용하기 위한 노력, SSD를 data processor로 확장하려는 노력에 대해 차례대로 설명한다. 결론에서는 간단한 요약과 함께 향후 Smart SSD의 발전 방향에 대해 논의한다.

Smart SSD는, 기존의 SSD가 제공하지 않는 새로운 기능을 제공함으로써 성능/수명/비용 관점에서 좀 더 효율적으로 동작하는 SSD 또는, SSD가 제공하는 새로운 특성들을 효율적으로 이용하는 host SW system을 의미한다.

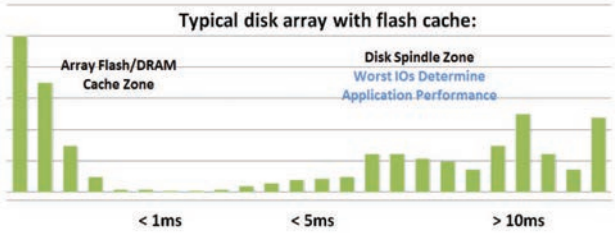
Ⅱ. 본 론

1. 예측 가능한 저장장치로서의 Smart SSD

SSD가 빠른 성능을 보여주는 것은 SSD 내부에 있는 다수의 NAND chip들이 동시에 동작하기 때문이다. 예를 들어, NAND chip 하나의 쓰기 성능은 20MB/s 정도이지만, 64개의 NAND chip이 있는 SSD는 이론적으로 최대 1280MB/s의 쓰기 성능을 낼 수 있다. 이런 높은 수치는 HDD로는 따라오기 어려운 (특히 random read/write의 경우) 높은 throughput 이다. 하지만, storage는 throughput 뿐만 아니라 latency 관점에서도 일정한 수준의 성능을 제공하는 것이 중요하다. 예를 들어, 어떤 저장장치의 초당 read 성능이 1024MB/s 정도로 매우 높다고 하더라도 각 read request의 처리 시간이 수백 us ~ 수십 ms 사이의 예측 불가능한 값이라면 사용자는 가끔씩 long latency로 인한 시스템 freezing 현상을 경험하게 될 것이다. 특히, web 검색 엔진처럼 web browser를 통해서 사용자가 체감하는 latency는 중간의 local OS, internet, web server, 원격 storage network, DB storage 등 많은 layer를 거치면서 누적되기 때문에 최하단인 SSD에서의 long latency는 전체 누적 latency에 더욱 악영향을 끼칠 수 있으며 사용자의 정보 접근성을 떨어뜨림으로써 Internet business에 악영향을 줄 수 있다[1].

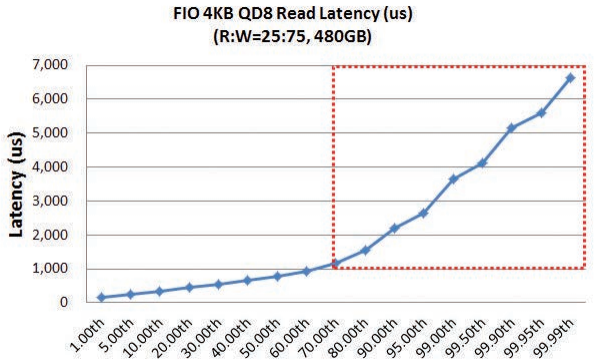
1.1 SSD 기반 Storage의 요구 Latency

〈그림 1〉은 flash storage를 cache로 사용하는 HDD 기반 storage 시스템에서의 IO latency 분포를 나타낸 것이다*. DRAM이나 flash storage의 cache에서 hit 된 request는 1ms 이내의 짧은 latency를 보이는 반면, cache miss로 인해 HDD에서 읽어야 하는 데이터의 latency는 5ms를 넘는 것을 볼 수 있다. 그렇다면, HDD를 NAND flash 기반의 SSD로 변경하면 위와 같은 5ms 이상의 long latency가 사라질 것인지 궁금할 것이다. 결론부터 얘기하면 그렇지 않다.



〈그림 1〉 Storage system에서 IO latency 분포의 예

* http://www.purestorage.com/flash-array/performance.html



〈그림 2〉 SATA SSD의 READ latency 누적분포함수 그래프

〈그림 2〉는 최신 SATA III SSD 상에서 FIO** benchmark tool을 사용해서 queue depth 8로 4KB random read와 write를 25:75 비율로 수행했을 때의 IO latency 분포를 누적분포함수로 표현한 것이다. X축은 누적 확률이고, Y축은 IO latency이다. 그래프에서 볼 수 있듯이 특히 read latency의 경우 약 70%를 넘어서면서 (즉, 약 30%의) read latency가 1ms를 넘는 것을 확인할 수 있다. 이 값은 write가 많아지면 더 악화될 수 있다. 요즘 All Flash Array 기반 storage system들이 read latency를 1ms 이하로 관리하려는 것을 볼 때 1ms 이상의 latency는 이슈가 될 수 있다.

1.2 SSD의 Latency가 불규칙한 이유

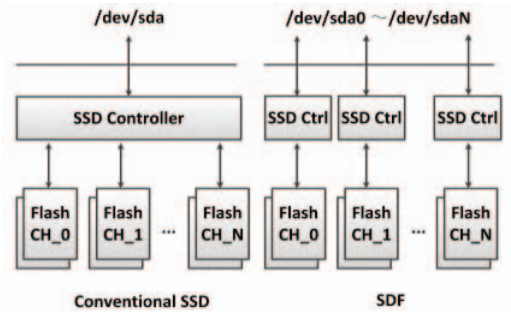
MLC 기반 NAND flash의 read time이 보통 60us 이내이고 write time이 1.5ms 이내임을 고려하면 위와 같은 long latency는 이해하기 어려운 값이다. 이것은 SSD를 구성하는 NAND flash의 특성에 기인한다. 하나의 NAND flash chip은 여러 개의 NAND block으로 나뉘고, 다시 하나의 NAND block은 여러 개의 page로 나뉜다. NAND에 데이터를 쓰려면 block 단위로 erase를 먼저 해야 하며, 데이터를 쓰거나 읽을 때에는 page 단위로 동작한다. NAND에 있는 특정 block 또는 page에 대해서 erase나 write 또는 read 동작을 하는 동안 같은 NAND flash chip에 속하는 다른 block이나 page는 접근할 수 없다. erase의 수행 시간은 보통 5ms 이하이지만 이 값은 SSD를 사용함에 따라 더 길어질 수 있다. erase를 막 시작한 시점에 해당 NAND flash의 다른 block에 속하는 데이터에 대한 read 요청이 도착하였다면, 이 read 요청은 erase 동작을 완료한 이후에야 진행할 수 있다. 즉, 5ms + 60us 정도의 latency가 예상된다. 문제는 host의 입장에서 SSD가 언제 이런 동작을 할 지 모른다는 것이다. 따라서, host는 60us~6ms 사이의 임의의 latency를 겪게 된다. Host의 입장에서는 latency가 좀 길더라도 균등한 latency를 선호하기 때문에 위와 같은 전형적인 SSD의 특성은 server향 SSD 같이 균등한 latency를 기대하는 system에서 약점이 될 수 있다. 이것이 바로 SSD의 latency 불규칙성 이슈이다[2-3].

> Host가 SSD의 IO latency에 대한 예측을 가능하게 하는 방법으로 host가 SSD 내부에서 일어나는 동작을 관리하는 것을 생각할 수 잇다.

1.3 SSD의 예측 가능성 향상 방안

Host가 SSD의 IO latency에 대한 예측을 가능하게 하려면 뭔가 변화가 필요하다. 이에 대한 해결방법으로 host가 SSD 내부에서 일어나는 동작을 관리하는 것을 생각할 수 있다. SSD는 내부 동작을 제어하는 새로운 기능을 host에 제공하고, host는 자신이 원하는 시점에 SSD의 각 동작을 수행시키며, 각 동작이 수행되는데 걸리는 평균시간을 고려하여 전체적인 IO를 스케줄링 할 수 있다. 이러한 approach의 극단적인 예가 〈그림 3〉에서 소개된 SDF (Software Defined Flash) SSD 이다[4]. SDF는 PCIe interface를 지원하는 고성능 SSD로써 SSD 내부의 병렬 동작 구조를 host에 노출하고 erase 역시 host가 제어하게 함으로써 host가 자신이 수행하려는 작업의 동작시간을 예측할 수 있게 하였다. SDF는 44개의 channel을 제공하고, 각 channel에는 8GB NAND flash chip 2개가 연결되는 구조인데, 각 channel에 대한 erase와 write는 8MB

** http://freecode.com/projects/fio



〈그림 3〉 SSD의 내부 동작 구조를 호스트에 공개하는 예

단위로, read는 8KB 단위로 하도록 제한함으로써 SDF 내부의 parallel component를 host로 완전히 노출시키고 전체 storage 공간을 host에게 제공하여 host가 SSD 내부에서 수행해야 할 작업을 모두 제어할 수 있다.

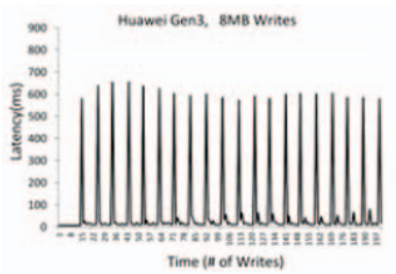
〈그림 3〉은 SDF의 각 channel에 연결된 NAND flash chip들이 어떻게 host system에서 접근되는지 보이고 있다. Host는 /dev/sdaX device file을 통해서 각 channel에 연결되어 있는 NAND chip을 독립적으로 제어할 수 있다.

SSD의 내부 동작을 host에서 제어함으로써 host는 거의 일관된 latency time을 얻을 수 있다. 〈그림 4, 5〉는 storage의 전체 영역을 거의 채운 후 8MB의 데이터를 write할 때의 latency를 측정한 것이다[4]. 〈그림 4〉의 일반적인 SSD에서는 7ms~650ms 사이의 예측 불가능한 latency를 보이지만 〈그림 5〉의 SDF에서는 383ms의 꾸준한 latency를 보인다. Host의 입장에서는 average latency가 73ms인 기존 SSD 보다는 383ms인 우측을 선호한다고 한다. 사실, SDF의 경우 44개의 channel에 대한 write를 parallel하게 독립적으로 처리하기 때문에 8MB 쓰기를 44개의 각 channel별로 동시에 issue 하더라도 약 383ms의 latency를 보인다. 반면, 기존의 경우 2000ms~3500ms 사이의 random한 latency를 보인다.

이런 방식의 추가적인 장점은 life time이 같거나 유사한 속성을 가지는 data를 서로 다른 NAND block에 분리하여 기록할 수 있다는 것이다. 이를 통해 전체적인 garbage collection의 cost를 줄이고 NAND의 수명을 연장할 수 있다. SDF는 erase와 write size가 8MB이고, 44개의 독립적인 worker thread가 있는 특정한 응용에 최적화 되어 설계된 SSD이며, Baidu의 CCDB라는 특정 응용에서 위와 같은 동작방식과 latency가 허용되기 때문에 유용하게 사용될 수 있었다[4].

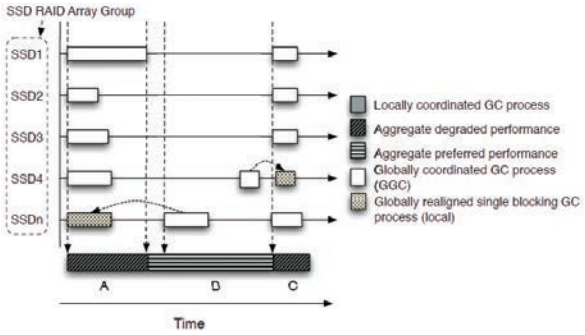
이렇게 각 SSD의 내부 동작을 제어할 수 있으면 2개 이상의 SSD로 구성된 storage array system에서는 각 SSD의 동작을 global하게 scheduling 함으로써 보다 더 예측 가능한 storage system을 구축할 수 있다[2-3]. 〈그림 6〉은 SSD array에서 각 SSD가 garbage collection 동작을 수행하는 순간을 일치시킴으로써 전체 storage system level에서 관측되는 long latency 구간을 최소화 하는 방법을 제안하고 있다[2]. 만약 data가 SSD array 상에서 2군데 이상에 저장되어 있다면 (또는 redundancy를 통해

각 SSD의 내부 동작을 제어할 수 있으면 두개 이상의 SSD로 구성된 atorage array system에서는 각 SSD의 동작을 global하게 scheduling 함으로써 전체적으로 보다 더 예측 가능한 storage system을 구축할 수 잇다.



〈그림 4〉 기존 PCIe SSD에서 8MB 쓰기 latency

〈그림 5〉 SDF PCIe SSD에서 8MB 쓰기 latency



〈그림 6〉 Global GC 스케줄링을 통한 latency 제어의 예

data를 읽을 수 있는 방법이 2가지 이상이라면), 하나의 SSD에서 garbage collection을 하는 동안 다른 SSD들은 read를 처리하는 방법으로 일정한 read latency를 보장하는 global scheduling도 가능하다[2]. 이러한 global scheduling은 SSD의 내부 동작을 제어할 수 있는 새로운 interface 명령을 SSD가 host에 제공함으로써 운영이 가능하다.

2. RAM 확장으로서의 Smart SSD

빠른 SSD를 DRAM인 것처럼 활용하려는 노력이 있다. 이는 처리해야 할 데이터의 크기는 수백 GB로 크고 locality는 적은 big data analysis나 key-value 응용에서 유용하다. 이런 응용에서 SSD에 저장되는 데이터는 휘발성이어도 되지만 IO latency는 매우 짧아야 한다. 최근의 PCIe interface는 memory access 방식으로 storage에 접근하는 것이 가능하며, 직접 DRAM interface로 연결하는 NVDIMM 타입의 SSD도 등장하였다*.

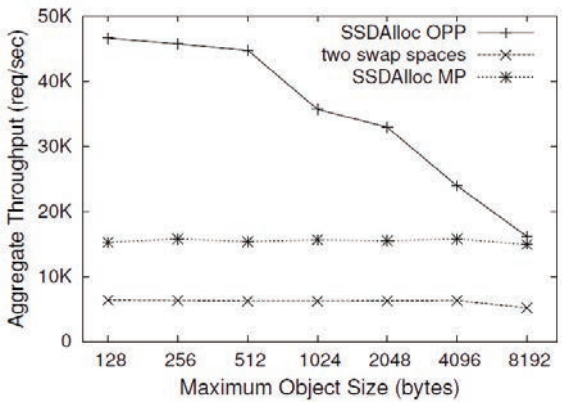
2.1 SWAP 최적화를 통한 DRAM 확장

SSD로 DRAM을 확장하는 가장 쉬운 방법은 swap device로 SSD를 사용하는 것이다. OS가 제공하는 virtual address 공간의 크기를 실제 host에 설치되어 있는 물리적인 DRAM 보다 크게 설정하고, 부족한 메모리를 SSD로 swap out & in 하는 방식으로 마치 큰 크기의 DRAM이 host에 있는 것처럼 동작한다. 이 방법은 응용프로그램이나 OS의 변경 없이 SSD를 swap 용 device로 활용할 수 있다는 장점이 있으며, write 비용이 매우 큰 SSD의 특성을 고려하여 SSD로의 write가 최대한 적게 발생하도록 page cache를 운영하는 방법으로 추가적인 최적화가 가능하다[5]. 또한, SSD 내부적으로도 swap용 IO pattern에 최적화 하여 FTL의 운영 정책을 결정할 수 있다.

빠른 SSD를 DRAM인 것처럼 활용하려는 노력이 있다.

2.2 Heap Allocator Library를 이용한 DRAM 확장

Heap memory allocator library에서 사용자 data와 SSD 사이의 DATA out & in을 SSD의 특성을 고려한 효율적인 방법으로 처리하는 기법이 SSDAlloc이라는 이름으로 제안되었다[6]. 이 방법은 응용프로그램에서 동적으로 할당하여 사용하는 memory data의 크기가 4KB보다 훨씬 작은 것에 착안하여 각 memory object의 크기에 딱 맞게 memory를 할당하고, SSD로 swap out을 할 때에는 여러 object들을 묶어서 큰 덩어리로 저장하는 방식으로 physical DRAM의 효율성과 SSD로의 IO 효율성을 높인다. 이 방법은 약간의 application 변경이 필요하지만 주로 처리하는 데이터의 크기가 4KB 보다 작은 경우 앞에서 설명한 swap 방식보다 더욱 효율적일 수 있다[6]. 〈그림 7〉에 성능평가 그래프가 있다.



〈그림 7〉 Object 크기에 따른 throughput 변화 (50% get)

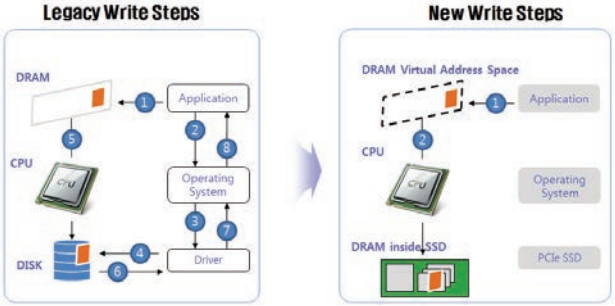
* http://en.wikipedia.org/wiki/NVDIMM

Object size가 작을 수록 효과가 큰 것을 알 수 있다.

2.3 PCIe를 이용한 DRAM 확장

SSD의 Interface가 SATA, SAS에서 PCIe/NVMe로 진화하면서 interface latency가 감소하고 bandwidth가 크게 증가하였다. 또한, PCIe 기반 SSD에서는 host에서 SSD 내부의 bus address 공간을 memory operation으로 접근할 수 있다[7-8].

〈그림 8〉은 PCIe 기반 SSD에서 어떻게 application이 SSD에 data를 전달할 수 있는지 표현한 것이다[8]. 좌측은 전통적인 방식으로 OS와 device driver를 통하는 것이고, 우측은 SSD에 있는 DRAM 영역을 host의 memory address 공간에 mapping 한 후 memory access 방식으로 접근하는 것이다. 기존 방식은 각 단계마다 SW가 개입하여 IO latency를 유발했지만 우측은 초기에 address mapping만 하고 나면, 이후부터는 SW의 개입 없이 IO가 처리되어 전체적인 IO latency 단축에서 큰 이득이 발생한다. 또한, 64-byte 단위의 접근이 가능하기 때문에 꼭 필요한 크기만큼만 IO를 할 수 있어서 workload에 따라 추가적인 성능 이득을 기대할 수 있다. 이러한 memory access 방식의 IO path를 활용하여 새로운 기능들을 추가할 수 있는데, DRAM을 확장하는 "Extended Memory", volatile 이지만 특정 시점의 상태에 대한 snapshot을 제공하는 "Checkpointed Memory", 비휘발성 특성을 지원하는 "Auto-Commit MemoryTM" 등의 응용이 제안되었다[8]. SNIA의 NVM Programming Technical Work Group에서는 Persistent Memory Device를 고려한 API를 표준화 하고 있는데, 전통적인 file command를 통한 IO 뿐만 아니라 load/store 기반의 memory access를 통한 IO까지 같이 정의하고 있다[10-13]. 향후 NVMe SSD가 일반화 되면 NVM Programming Model을 활용한 application들이 출현하게 될 것이다.



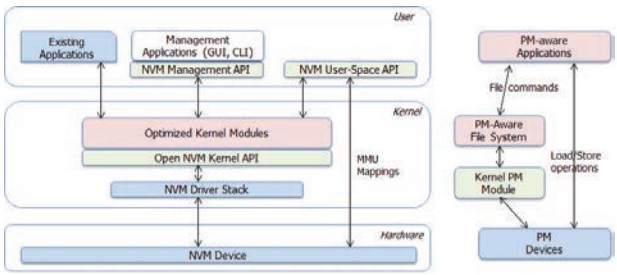
〈그림 8〉 PCIe 기반에서 SSD에서 Direct IO Path

3. CPU의 확장으로서의 Smart SSD

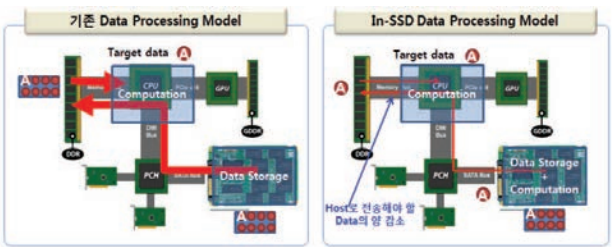
SSD의 또 다른 특성은 computing power를 가지고 있다는 것이다. 고성능 IO를 지원하기 위해서 SSD 내부에 여러 개의 CPU를 높은 clock으로 운영하는 추세이다. 하지만, SSD에서 일반적인 CPU 연산을 모두 처리하기는 어렵다. 대신, 처리해야 하는 data의 양은 매우 크고 (SSD당 수백 GB 이상), 처리 결과는 매우 작은 (수십 MB 이하) 경우 SSD에서 data를 처리하는 것이 system 전체적으로 성능과 에너지 면에서 유리할 수 있다[14-15,18].

SSD의 또 다른 특성은 computing power를 가지고 있다는 것이다.

〈그림 10〉은 SSD 안에서 data를 처리하는 경우 예상되는 이점을 표현한 것이다. 붉은 색 화살표는 SSD



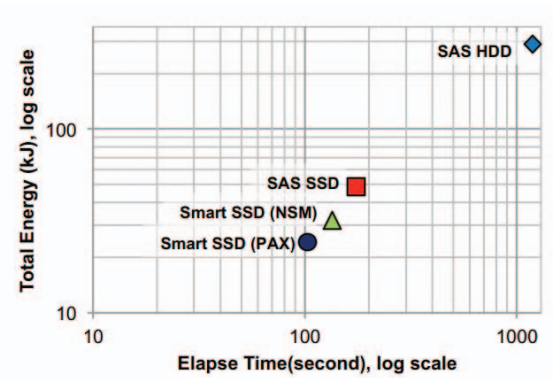
〈그림 9〉 SNIA NVM Programming Model



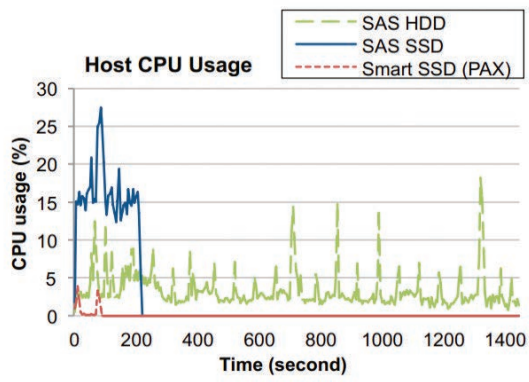
〈그림 10〉 In-SSD Data Processing Model

에서 host로 이동하는 data를 의미한다. 'A'는 분석 결과를 나타낸다. In-SSD data processing으로 얻을 수 있는 첫 번째 장점은 data 전송 시간의 단축과 에너지의 사용 감소이다. 이것은 host로 전송해야 할 data의 크기가 크게 감소하기 때문에 기대되는 이점이다. 두 번째 장점은 host CPU 대신 SSD 내부의 computing power를 사용함으로써 host CPU의 비용을 줄이거나 host CPU를 다른 용도로 활용할 수 있다는 것이다. Data 분석 process 중에 어떤 기능을 SSD 안에서 수행하는 것이 전체적으로 가장 최적화 된 것인가는 응용에 따라 다를 수 있다. [15]에서는 Microsoft SQL Server의 query processing 관련 일부 기능을 SSD 안에서 처리하도록 구현하였다. 〈그림 11〉은 TPC-H query 6을 수행한 경우 얻을 수 있는 성능과 에너지 관점에서의 개선사항을 나타낸 것이다[15]. SAS SSD와 대비하여 시간은 1.7x, 에너지는 2.x배 개선된 것을 확인할 수 있다. 여기서 NSM (N-ary Storage Model)과 PAX는 DB page에서 각 record를 저장하는 방법을 나타내며, selectivity factor란 전체 data의 크기 대비 처리 결과 데이터의 크기 비율을 의미한다.

SSD는 HDD를 대체하는 데 그치지 않고 새로운 기능을 제공함으로써 더 발전된 저장장치로의 변화를 모색하고 있다.



〈그림 11〉 TPC-H query 6 on the LINEITEM table (0.6% selectivity factor)



〈그림 12〉 SAS HDD/SSD/Smart SSD별 Host CPU 사용율

〈그림 12〉는 동일한 작업을 SAS HDD/SSD/Smart SSD에서 각각 수행했을 때 host CPU의 사용율을 시간에 따라 나타낸 것이다[15]. In-SSD Computing을 적용한 경우가 시간도 짧게 걸리고 CPU 사용율도 적음을 확인할 수 있다.

[14,15,17]의 연구가 SSD 내부에 있는 기존의 ARM CPU에서 data processing을 하는 방식이었다면, [16,18]은 HW accelerator로 중요 data 처리 알고리즘을 구현하고, 이를 조합하여 data 분석을 수행하는 방식으로 성능을 훨씬 더 개선하고 에너지 소모도 더 크게 줄이고 있다. [14-18]의 동작방식은 전체 data 분석 프로그램 중 SSD 안에서 수행할 code는 이미 SSD에 구현되어 있고 host는 parameter만 전달하는 방식을 취하고 있는데, 향후 좀 더 flexible 한 program이 가능하도록 run-time에 host로부터 code까지 down 받아 실행하는 방식으로 발전될 수 있을 것이다.

Ⅲ. 결 론

SSD는 HDD를 대체하는데 그치지 않고 새로운 기능을 제공함으로써 더 발전된 저장장치로의 변화를 모색하고 있다. NAND flash로 구성된 SSD의 약점을 극복하기 위해 내부 동작의 제어권을 host로 옮기기도 하고, PCIe bus interface를 이용하여 load/store의 Memory access 방식에 기반한 새로운 기능을 제공하기도 하며, SSD 내부에서 데이터 분석을 처리하기도 한다. 본 논문에서는 이러한 시도가 특정 응용에 있어

서 높은 성능 및 수명 개선을 보일 수 있음을 살펴보았다. 이런 변화는 host에서도 SSD가 제공하는 새로운 기능을 smart 하게 이용할 수 있도록 host SW의 변경 및 최적화를 필요로 하는데, 앞으로 많은 연구가 필요한 부분이다[9].

전통적인 write/read 기능을 제공하는 것만으로는 SSD의 제품 차별화가 어렵기 때문에 위와 같은 새로운 기능을 제공하려는 시도는 앞으로도 계속될 것이며, 기존 연구자들도 위와 같은 관점에서 어떻게 SSD를 활용할 수 있을 지 지속적인 고민이 필요하다. SSD가 host의 필요에 따라서 새로운 기능을 제공할 수 있다는 관점에서 '어떻게 SSD가 host와 협력하여 전체적인 시스템 최적화를 할 수 있을까' 앞으로 무궁무진한 SSD의 발전이 기대된다.

참 고 문 헌

[1] E. Schurman, J. Brutlag, "The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search," O'Reilly Velocity Web Performance and Operations Conference, San Jose, CA, June 2009, http://velocityconf.com/velocity2009/public/schedule/detail/8523.

[2] Y.J. Kim, S. Oral, G.M. Shipman, J.H. Lee, D.A. Dillow, F. Wang, "Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-state Drives," IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), pages 23-27, Denver, CO, USA, 2011.

[3] S.C. Miller, J.S. Kimmel, "Method and apparatus for achieving consistent read latency from an array of solid-state storage devices," US 8021142 B1, http://www.google.com/patents/US8621142, 2013.

[4] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, "Software-defined flash for web-scale internet storage systems," ASPLOS 14 Proceedings of the 19th Int'l conference on Architectural support for programming languages and operating systems, pages 471-484. Salt Lake City, UT, USA, March 2014.

[5] M. Lin, S. Chen, G. Wang, "Greedy page replacement algorithm for flash-aware swap system," IEEE Transactions on Consumer Electronics, Vol. 58, Issue 2, pages 435-440, May, 2012.

[6] A. Badam, V.S. Pai, "SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy," 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, USA, March 2011.

[7] L. Wang, "How to Implement a 64B PCIe Burst Transfer on Intel Architecture," Intel White Paper, Feb. 2013.http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/pcie-burst-transfer-paper.pdf

[8] B. Compton, "Auto Commit Memory: Cutting Latency by Eliminating Block I/O," Jan. 2012, http://www.fusionio.com/blog/auto-commit-memory-cutting-latency-by-eliminating-block-i/o/

[9] X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, D.K. Panda, "Beyond Block I/O: Rethinking Traditional Storage Primitives," 2011 IEEE 17th Int'l Symposium on High Performance Computer Architecture (HPCA), pages 301-311. San Antonio, TX, USA, Feb. 2011.

[10] N.Talagala, "NVM Software Interfaces New Directions," SNIA NVM Summit, Feb. 2013. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130814_Keynote5_Doller.pdf

[11] A. Rudoff, "Programming Models to Enable Persistent Memory," Storage Developer Conference, Santa Clara, 2012.http://www.snia.org/sites/default/files2/SDC2012/presentations/Solid_State/AndyRudoff_Program_Models.pdf

[12] A.Rudoff, "The Impact of the NVM Programming Model," Storage Developer Conference, Santa Clara, 2013.

[13] SNIA, "NVM Programming Model (NPM) Version 1," SNIA document, Dec. 21, 2013.

[14] Y.W. Kang, Y.S. Kee, E.L. Miller, C. Park, "Enabling cost-effective data processing with smart SSD," IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), pages 1-12, Long Beach, CA, USA, May 2013.

[15] J. Do, Y.S. Kee, J.M. Patel, C. Park, K. Park, D.J. DeWitt, "Query processing on Smart SSDs: opportunities and challenges," Proceedings of the 2013 ACM SIGMOD Int'l Conference on Management of Data, pages 1221-1230, New York, NY, USA, 2013.

[16] J. Ouyang, S. Lin, Z. Hou, P. Wang, Y. Wang, G. Sun, "Active SSD Design for Energy-efficiency Improvement of Web-scale Data Analysis," IEEE Int'l Symposium on Low Power Electronics and Design (LPED), pages 286-291, Beijing, China, 2013.

[17] E.M. Doller, "Storage Architecture for the Data Center in 2020," Flash Memory Summit 2013.

[18] S.Y. Cho, C.I. Park, H.O. Oh, S.C. Kim, Y.M. Yi, G. Ganger, "Active Disk Meets Flash: A Case for Intelligent SSDs," Proceedings of the ACM Int'l Conference on Supercomputing (ICS), pages 91-102, Eugene, OR, June 2013.

권 문 상

1995년 서울대학교 컴퓨터공학과 학사 졸업
1997년 서울대학교 컴퓨터공학과 석사 졸업
2003년 서울대학교 컴퓨터공학과 박사 졸업
2003년~2006년 삼성전자 SW센타
2006년~현재 삼성전자 반도체

〈관심분야〉
OS, System SW, Flash Storage, Security, Cryptography, System Optimization