Ziting Liu

Shiyue Hou

EECE5643

NVMe SSD Emulator test with FIO

Introduction

Because of SSDs' high throughput, low latency and lowered costs over time, SSDs have become mainstream in many Cloud Enterprise, such as Amazon AWS and Google Cloud, as well as consumer devices such as laptop, workstations. As the demand for better I/O performance increases at cloud enterprise, data center and consumer applications, conventional SSD architecture can no longer keep up.

As SSDs and their associated protocol develop, the industry needs an accurate and stable emulator to emulate the SSDs characteristics. In our project, we will be using FEMU to perform simulations. FEMU is an emulator based on the QEMU/KVM virtual machine. It can accurately emulate SSDs with multiple SSD types. Customers can choose the SSDs type according to their research requests. Moreover, FEMU is an open-source emulator software provided for free. FEMU also has great scalability, supporting up to 32 channels and chips. In our project, we will simulate the performance of I/O operations on an SSD. Specifically, we will try to figure out what are the most important factors that will impact the performance of a SSD, as well as less important factors that are not as important as we thought.

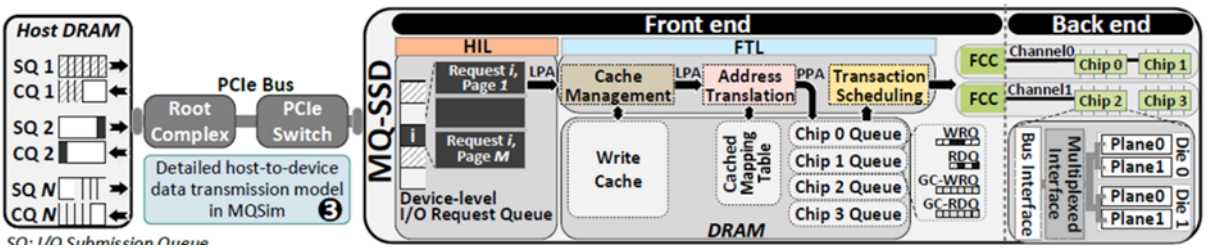

Figure 1: Architecture of SSD [2]

Background on SSD Architecture

Solid state drive (SSD) is a storage device that uses integrated circuit assemblies to store data. Unlike traditional HDD, it has no moving parts. There are two key components in an SSD: controller and memory. Controllers are one of the most important factors that affects SSD performance. SSD controllers include electronics that bridge the flash memory components to the SSD input/output interfaces. There are two major memory types: Flash and DRAM. DRAM memory has advantages in terms of performance but usually costs more than NAND flash memory. Other than the two components mentioned above, there are Cache, Capacitor, and Host Interface. A capacitor, or a battery, is used to ensure data integrity when power is cut off. Finally, Host Interface is the physical connector managed by the controller, there are many different interface protocols.

SSDs typically consist of two parts, Front End, and the Back End. Front End includes a Host-Interface Logic (HIL) and a Flash Translation Layer (FTL). The Back End are the physical storage components.

Host-Interface Logic (HIL) plays a very important role in utilizing NAND flash memory parallel to improve SSDs throughput, IOPS and bandwidth, as well as decrease response time. The SATA protocol is generally used in traditional SSDs. SATA adopts Native Command Queue, which supports parallel execution IO requests based on SSDs space available. NVMe is designed for SSDs. NVMe enables SSDs scalable, high throughput and bandwidth, lower response time within communication by PCIe bus. At Host DRAM, applications insert a job directly into the submission queue (SQ), SSDs select job form submission queue and perform IO operation and return operation result into completion queen (CQ). Modern SSDs have broadly adopted the NVMe protocol.

Flash Translation Layer (FTL) runs on a microprocessor inside SSDs. When SSDs select a job from the host's DRAM submission queue, HIL inserts the job into Device-Level IO Request Queue. FTL will check if the job operation type is read or write. For write operations, the job writes trigger FTL write cache management unit to store data and ask HIL to prepare a response. For read operation, FTL translates the job logic address into physical address to process. When the job finishes, FLT will ask HIT to send a response to the host. Wearing level and garbage collection also execute at FTL. Wearing level is a SSDs technique to increase SSDs lifetime. The principle of wear level is to evenly distribute write on all of

SSDs blocks. All cells received the same number of writes, this is to avoid writing too often in specific one. Garbage collection is an optimized machine to release invalid blocks.

When data is deleted from the file system, FTL marks erase pages as invalid, moving valid pages into a free block and then erases invalid blocks. The modern SSDs are produced using NAND flash memory chips. Inside the NAND flash memory chip, several pages grouped as a block. Data erases operate at block. Flash writes take place at pages. Figure 2 shows the structure of the backend. The back end includes multiple bus channel. The channel connects the frontend and backend. Each channel connects multiple chips. Each chip contains multiple die. Every die can independently perform memory commands. Each die separates into multiple planes. Each plane is divided into multiple blocks. Block made up multiple pages. One page default 4KB.

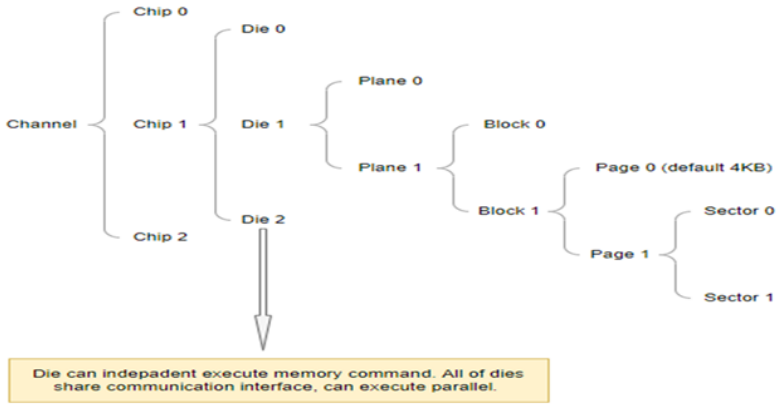

Figure 2: Structure of Back End

FEMU Introduction and Installation

FEMU is a NVMe SSD emulator based on QEMU/KVM. FEMU is exposed to Guest OS as a NVMe block device. On our machine we had Linux as our Host OS, inside our Host OS we had another Linux virtual machine installed as our Guest OS. In this Guest OS, FEMU is installed as an SSD device, we can perform I/O operations on this SSD as if it is a real SSD device. We choose FEMU because FEMU is a fast, scalable, and extensible SSD emulator.

Traditional SSD emulator uses image files to emulate the disk. When the system is doing I/O operation the VM has to exit to perform the operation. VM exits are expensive in terms of time. When the workload requires simulating multiple threaded I/O operations, the I/O latency grows exponentially whereas FEMU can still achieve a relatively low latency. The

way FEMU achieved this is by using DRAM instead of image files. FEMU allocated space in DRAM as a shared memory between Guest OS and FEMU to avoid expensive VM exits. For the accuracy model, FEMU used three different policies. First is called the Base Delay Model where every I/O operation has 50µs delay with an overhead of around 20µs. Above the Base Delay Model, FEMU has a Basic Delay Model used to calculate the scheduling such as arrival time, execution time, ready time, etc. Finally, FEMU supports OpenChannel SSD simulation where there are two registers per NAND gate to achieve better parallelism. FEMU is also very powerful in terms of Usability and Extensibility, but we will not be using too many different functionalities in our experiment.

In our experiment, we will be focusing on the Black Box mode. Black Box mode is the one with FTL managed by the device. This is the mode used for simulating most of the current commercial SSDs.

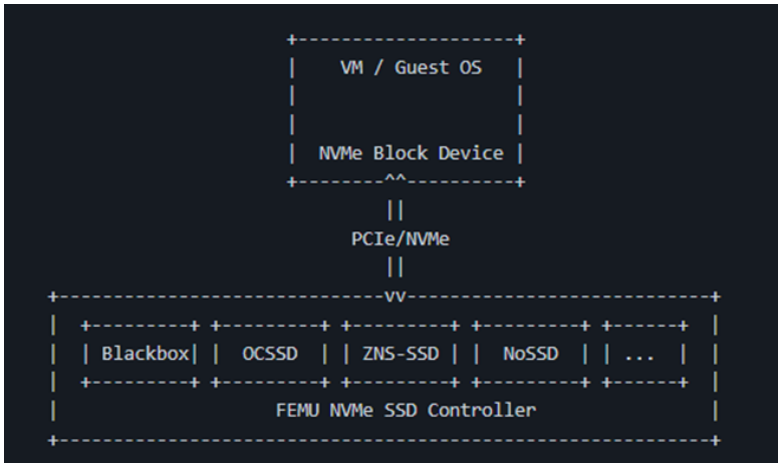

Figure 3: FEMU Architecture [1]

The installation process of FEMU was not as simple as we thought, especially when one of the group members never had any experience with Emulators before. Our first attempt was to install Linux in VirtualBox, then install FEMU inside our virtual Linux machine. Finally inside FEMU we will run our simulation softwares. This implementation did not work out, because for some reason our VirtualBox did not support KVM. This was not recommended anyways, because it is running a virtual machine inside a virtual machine. The performance is not going to be good even if it works. Our second attempt was to borrow a Linux server. The process did not run smoothly either, the first challenge we faced was not having enough privilege to run KVM nor sudo. After we solved the user privilege problem, we can finally

try to get FEMU working. The last difficulty we faced was that, FEMU requires a GUI environment for the installation process and our server has no graphical interface. This problem can be solved by redirecting VM output to console, otherwise the terminal will hang during the installation process.

Our conclusion on running FEMU can be concluded as the following:

- Recommended to use a server version Linux system
- Make sure to have sudo privilege to put yourself in KVM group, as well as to run sudo commands
- If graphics are not supported, use "-serial studio" to redirect output to console
- Delete "-localtime" if necessary

Simulation Process

In our experiment, we will be running two I/O benchmarks on FEMU, and we will be comparing our results with a real SSD device. The I/O benchmark we will be using is Fio. At first, our idea was to test the same workloads with different benchmark softwares. We looked into Bonnie++ and IOzone, but we realized that none of those softwares matches the functionalities of FIO, therefore we can not do a horizontal contrast with them. The real SSD we used in our experiment is Dell 512GB NVMe SSD with PCIe interface and M.2 form factor.

The goal of our experiment is to see what parameters will affect the performance of SSD the most. We will be able to find out what are the key factors that modern SSD devices should focus on. As well as what are the factors that are not as important as we would imagine. the parameters are described as follows:

- ioengine : How jobs issue I/O
- readwrite : I/O pattern
- numjobs : How many clone of this job
- size: Total size of file I/O
- thinktime: Stall between each job
- iodepth: The same is queue depth, Number of pending I/O request

Evaluation

For our first experiment, we set the size of the file as our variable. This is to find out how the size of I/O workload affects the performance of SSD. In our experiment, we tested the files with sizes from 200MB to 4GB. We can tell from Fig.4 and Fig.5 that varying the size of workload did very little effect on the performance. Although the specific numbers of FEMU and Real SSD are different, the trend was the same. We can conclude the size of workload will not affect the performance massively.

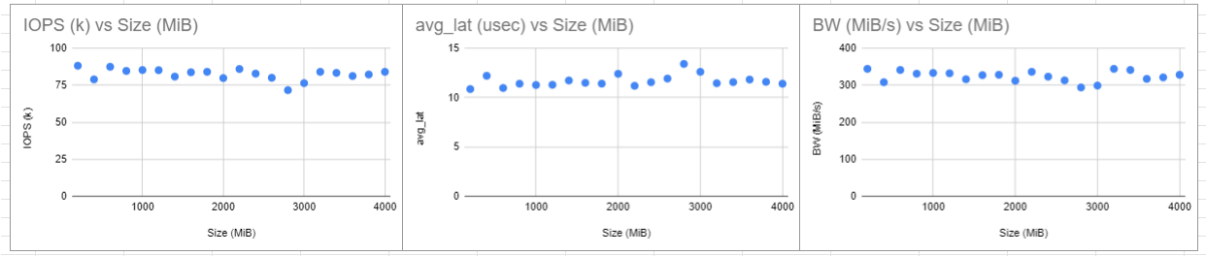

Figure 4: Results of Workload with different size with FEMU

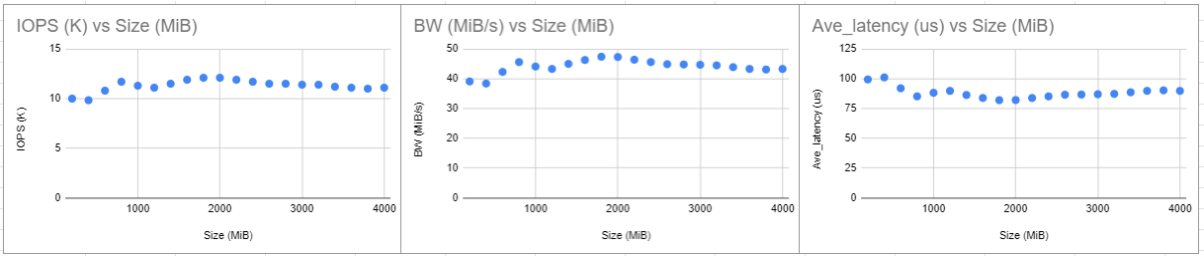

Figure 5: Results of Workload with different size with Real SSD

For our second experiment, we set thinktime as variable.This is to see how the delay time between each job affects the overall performance of an SSD. From Fig.6 and Fig.7 we can see that as thinktime increases, bandwidth and IOPS decreases exponentially. This change destroyed the performance of an SSD. The delay time between each I/O task is a critical factor that manufacturers should pay attention to. However an interesting observation is that although bandwidth and IOPS dropped exponentially, average latency did not grow much.

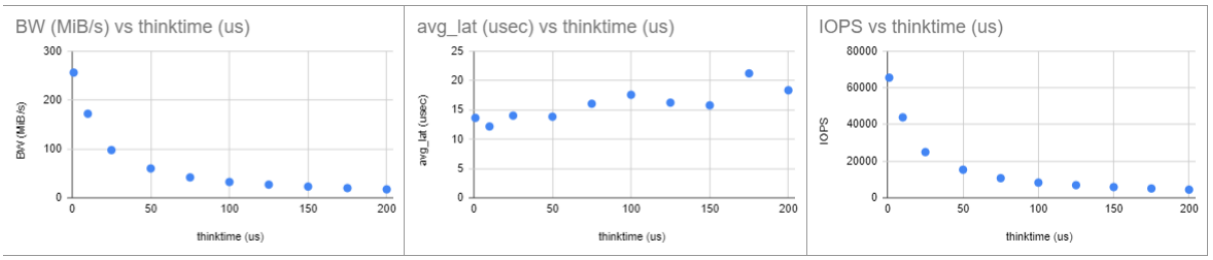

Figure 6: Results of Workload with different thinktime with FEMU

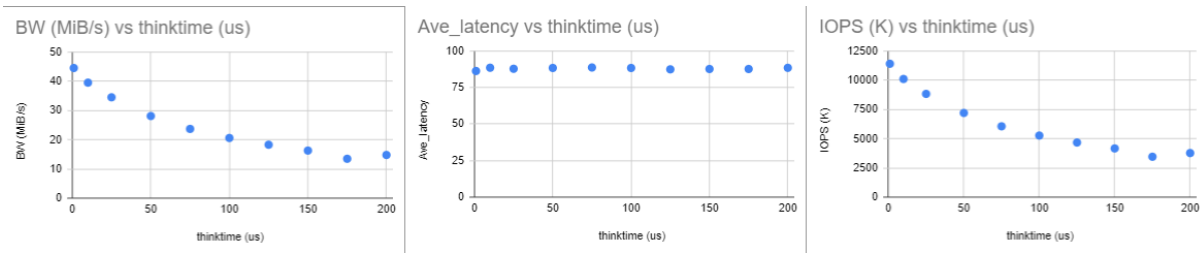

Figure 7: Results of Workload with different thinktime with Real SSD

For our third experiment, we set the number of jobs as the variable. Number of jobs is the number of threads executing the I/O task in parallel. People would imagine that parallel tasks should be as many as possible, but that is usually not the case. We will be finding out the balance point between cost and performance.As shown in Fig.8 and Fig.9. From the result we got from FEMU, we can see a peak in both IOPS and BW when the number of threads equals 8. Because the average latency in jobs grows as we use more threads. As the number of threads surpasses 8, the performance starts to decrease. However, this is not the case with real SSD. The performance of a real SSD increases linearly as we increase the number of threads. If we compare the numbers, we can see that average latency of a real SSD started from a relatively high number compared to FEMU. Although the average latency on real SSD is growing slowly, it did not affect the performance. Our conclusion was that the number of threads being used should be considered based on the average latency of the system. If the average latency of the system is low, then creating more threads will increase this number significantly, and cause the system to slow down. This is like running a simple computer program with 16 threads is slower than running on a single thread, the overhead of creating a thread is too expensive.

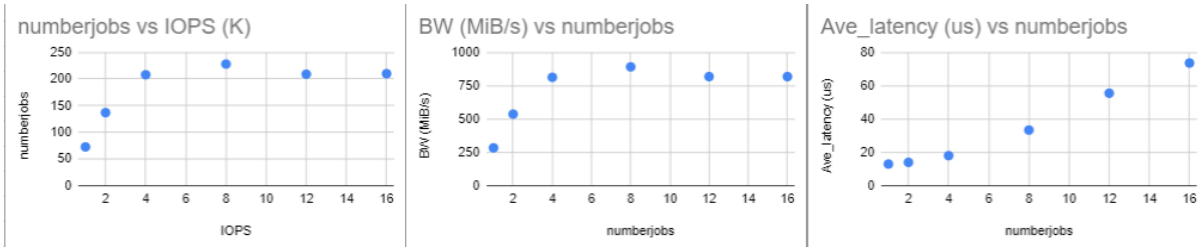

Figure 8: Results of Workload with different numjobs with FEMU

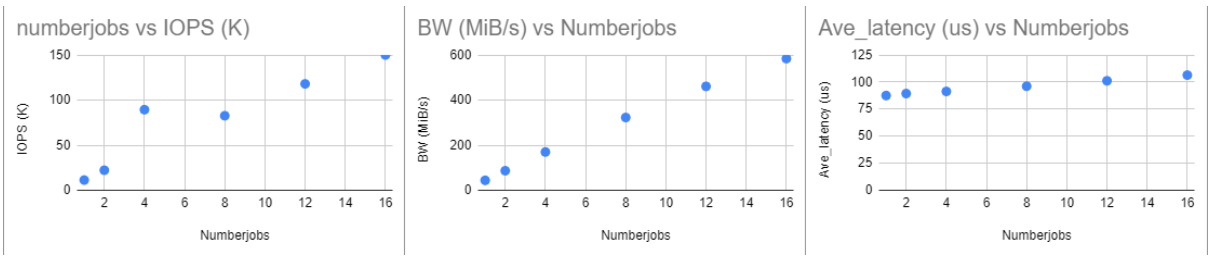

Figure 9: Results of Workload with different numjobs with Real SSD

For our last experiment, we set the iodepth as the variable. Iodepth is the same as queue depth, is the number of pending I/O jobs in the queue. Because iodepth will not affect the performance of a synchronized ioengine, we tested the workload on two different engines, psync and libaio (Linux native asynchronous I/O), and compared their differences. As shown in Fig.10 and Fig.11. We can see that iodepth did not affect the performance on synchronized systems. However in asynchronous ioengine we see different trends from FEMU and real SSD. In FEMU, the average latency increases as we increase the iodepth, and IOPS and BW stays at the same level. In the real SSD the average latency stayed the same while IOPS and BW increased.

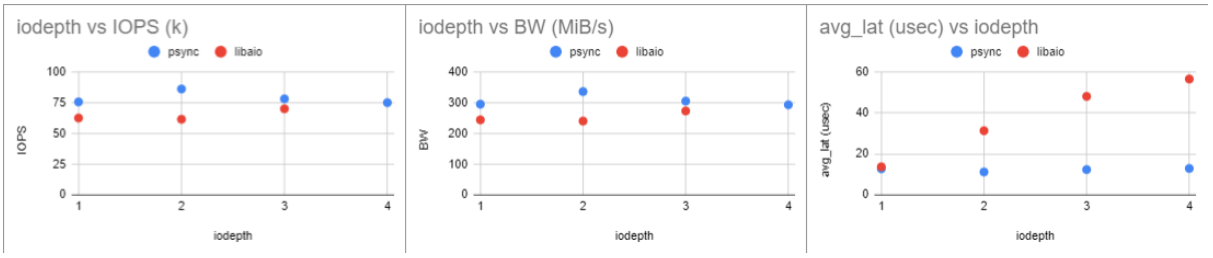

Figure 10: Results of Workload with different iodepth with FEMU

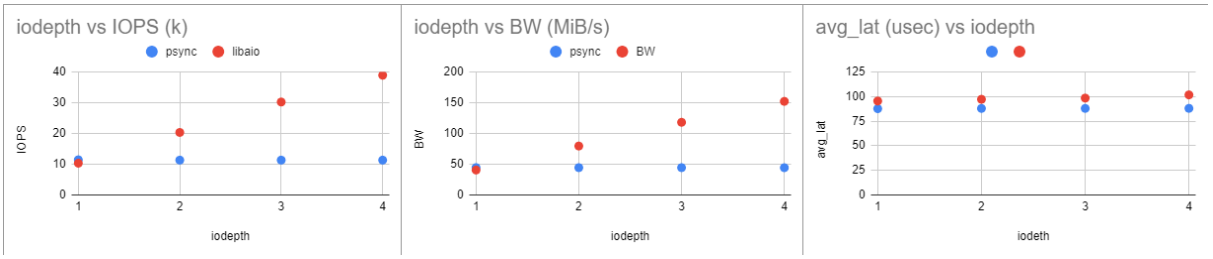

Figure 11: Results of Workload with different iodepth with Real SSD

Conclusion

In our project, we tested how Size of the workload, thinktime, iodepth, and number of jobs affect the performance of an SSD. The solution to making a fast SSD is not absolute. Designing such a system should be based on the hardware limitations of the device. Simplifying adding more of everything will result in worse performance.

Reference

1. Li, Huaicheng, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Bjørling, and Haryadi S. Gunawi. "The {CASE} of {FEMU}: Cheap, accurate, scalable and extensible flash emulator." In 16th USENIX Conference on File and Storage Technologies (FAST 18), pp. 83-90. 2018.
2. Tavakkol, Arash, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. "{MQSim}: A Framework for Enabling Realistic Studies of Modern {Multi-Queue}{SSD} Devices." In 16th USENIX Conference on File and Storage Technologies (FAST 18), pp. 49-66. 2018.
3. https://fio.readthedocs.io/en/latest/fio_doc.html