**INTRODUCTION AND MOTIVATION - PAST WORKS CHAPTER**

* QEMU+KVM vs Virtualbox - For Better Performance
* Current SSD Researches either support only internal-SSD Research and not kernel-level extensions  
  or  
  Full stack software/hardware research is currently done by paid softwares, very costly  
    
  Current cheap emulators are just not that good (non-scalable)
* This is the reason for need of FEMU. Features:
  + Open Sourced
  + Accurate (0.5-38% Variance wrt current LightNVM-QEMU platform)
  + Scalable with low latency
  + Extensible, internal-SSD as well as kernel-level
* Cost of research in Distributed SSD’s very high, 380 papers conferenced  
  Even using OpenChannel SSD during research will reduce life-expectancy of the SSD, so we need a software solution or an emulator  
    
  Moreover, exposure of internal channels and chips of OCSSD (via liblightnvm) is there, but the firmware logic is still not known that much, i.e. still a black box logic there, that’s why a so called ideal emulator design lacks.
* Current Emulators Drawbacks:
  + FlashEm - Based on Linux block level layer hence less portable
  + QEMU - cannot emulate multiple channels (IO Channels, maybe?), like OpenChannel SSD
  + VSSIM - not scalable, as built on QEMU Interface

**FEMU CHAPTER**

* FEMU - FTL and GC (Garbage Controller, maybe ?) not discussed
* **SCALABILITY:**ie with increasing number of threads, the average latency should not drastically increase, should support multi IO tasks in parallel, like in Open Channel SSDs

Two major problems faced by every emulator:

* + MMIO [Memory Mapped IO, MMIO uses same address space to store memory and IO-devices-registers] operation of IO will cause to switch from Guest OS to QEMU
  + Asynchronous IO done (which is necessary to avoid convoy effect due to more time consuming IO tasks), but AIO overheads are a concern when stored to RAM supported image (RAM’s are slower here)

Solutions:

* + Instead of Interrupt, Polling based design used for lesser MMIO switches (one extra QEMU thread running poll) *#one LOC change*
  + And, create their image in QEMU heap space, and DMA logic used here, i.e GuestOS doesn’t even knows about it and data transfers subsequently

Other Scalability problems:

* + QEMU single-thread event loop (performs the main IO routine such as dequeueing the device queue, triggering DMA emulations, and sending end-IO completions to the Guest OS)
* **ACCURACY:**
  + Delay emulation : T(endio) = T(entry) + 50us (but takes 20us more because of GuestOS overhead)
  + Aim is to calculate accurate T(endio) time for every IO
  + Plane is part of NAND Flash Arch:  
    *“For example, if a page write arrives to currently-free channel #1 and plane #2”*
  + Problem with current approach:  
    Only single page register, so write, then read takes place  
      
    Basic delay model is enough for FTL and GC comparison
  + **FEMU Advanced “OC” Delay** Model Approach:
    - *double-register* planes, consisting of data and cache registers
    - *non-uniform* page latency model, pages mapped to upper bits of MLC cells incur higher latencies than those mapped to lower bits
  + Results show a near accurate graph comparison between FEMU and OC-SSD. The graph has channels vs plane/channel axes. The basic model has high latency difference, the two changes of FEMU-advanced-OC model shows a higher accuracy, i.e. 0.5-38% , Tested on FileBench workload (which provides extensive application IO extensive Workload Model Language (WML)), kind of like sample scenarios EXata
  + What is Varmail part of FileBench used for Accuracy Charts where 38% error?
  + Advanced OC Model needs more CPU computations than Basic Model due to, as task division.
* **USABILITY AND EXTENSIBILITY (additional features of FEMU):**
  + **FTL and GC (Garbage Collector) schemes:**
    - FTL Dynamic Mapping
    - GC Channel Blocking, used over controller blocking (locks down controller preventing any IO, used by OC-SSD) and plane blocking (the best one).
  + Can be used as both WhiteBox SSD (where physical page addresses and FTL is managed by OS, like OCSSD) or Black Box SSD where OS knows only about the logical addresses.
  + Multi Device support, if FEMU exposes 4 SSDs, then inside 4 separate NVM (non volatile memory) instances and FTL’s are running, with non overlapping channels.
  + Extensible OS-to-SSD commands to avoid OS-to-DRAM-to-OC movement over bus, thus doing OS-to-OC directly.
  + **Distributed SSDs:**But how will they replicate SSD firmware black box efficiently on distributed N/W.
  + Page Level Fault Injection: by adding page level corruptions and faults and observing how high-level software stack reacts
* **LIMITATIONS:**
  + FEMU is DRAM backed, hence cannot emulate large capacity SSD.
  + Crash consistency for NVM (non volatile memory) testing needs to be done, like soft and hard reboots (How and why they implement NVM)?
  + Accuracy improvements needed (as discussed before)