Portfolio

Donghun Koo
amainlog@gmail.com
dhkoo.github.io

Research Experience

- **메인 연구**
  - 대규모 분산 시스템에서 고성능 I/O 처리를 위한 고성능 스토리지 시스템 개발 연구
  - 고집적 매니코어 시스템의 I/O 성능 특성 분석 및 최적 스토리지 시스템 연구
  - 클라우드 환경에서 이종저장장치를 활용한 적응형 하이브리드 스토리지 시스템 개발 연구

- **서브 연구**
  - 프라이빗 블록체인의 위험성 감지 및 알림 기능을 위한 python 기반의 로그 정보 분석 및 메일링 기능 데몬 개발
  - 데이터센터내의 빅데이터 워크로드를 위한 적응형 성능격리 시스템 개발
  - 빅데이터 처리 프레임워크 성능개선 연구

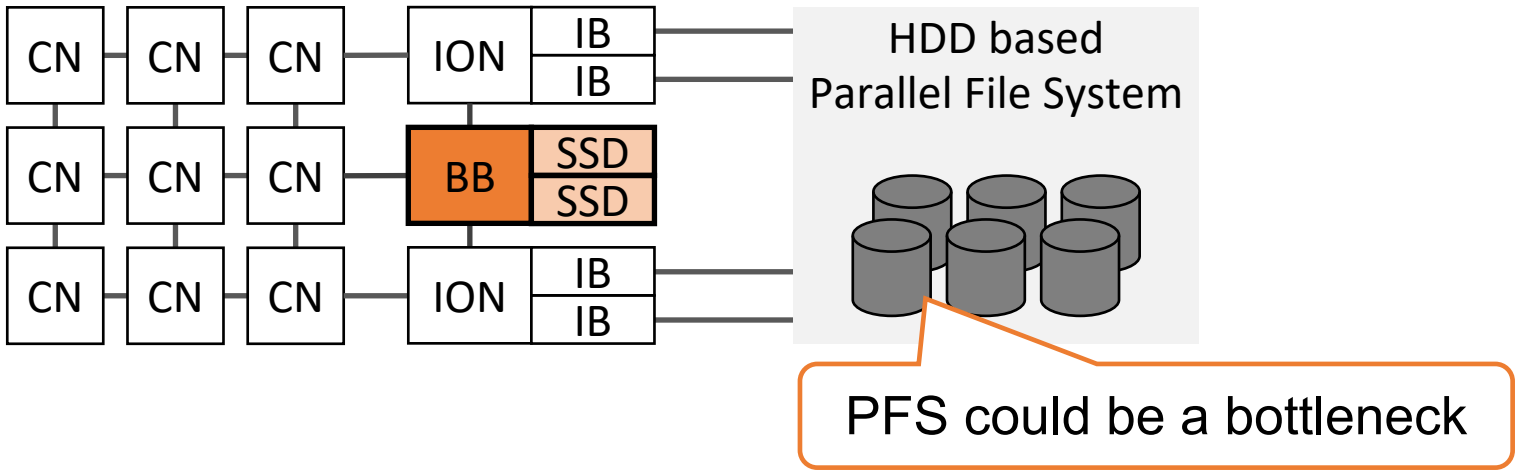
대규모 분산시스템에서 고성능 I/O 처리를 위한 고성능 스토리지 시스템

High-performance storage system to effectively handle the I/O in large-scale systems

Burst Buffer

- **Most supercomputers are using Hard Disk Drives (HDDs) based parallel file systems**
  - Burst Buffer technologies are being suggested to alleviate the performance issue using fast Solid-State Drives (SSDs)

- **Burst buffers could handle the bursty I/O effectively**
  - Reducing the overall application runtime

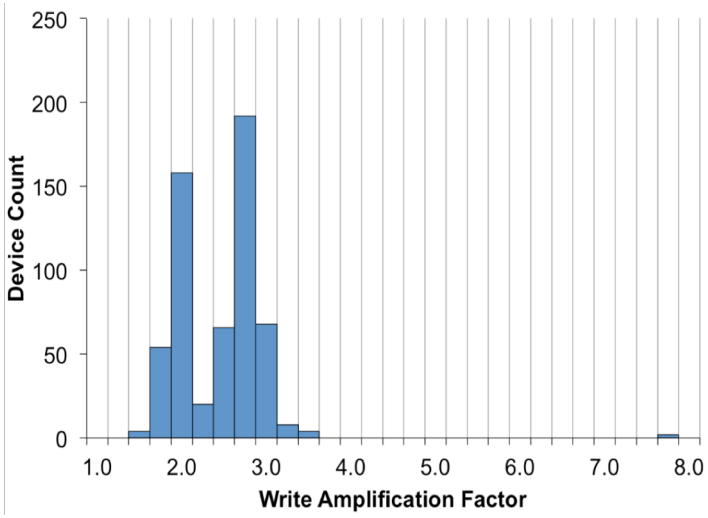


Problem Statement

- **SSDs are shared resources in Burst Buffers**

- **Supercomputing system performs a large number of I/O operations**
  - SSDs of burst buffer would be exposed to frequent GC operations, which can lead the losses of performance and endurance

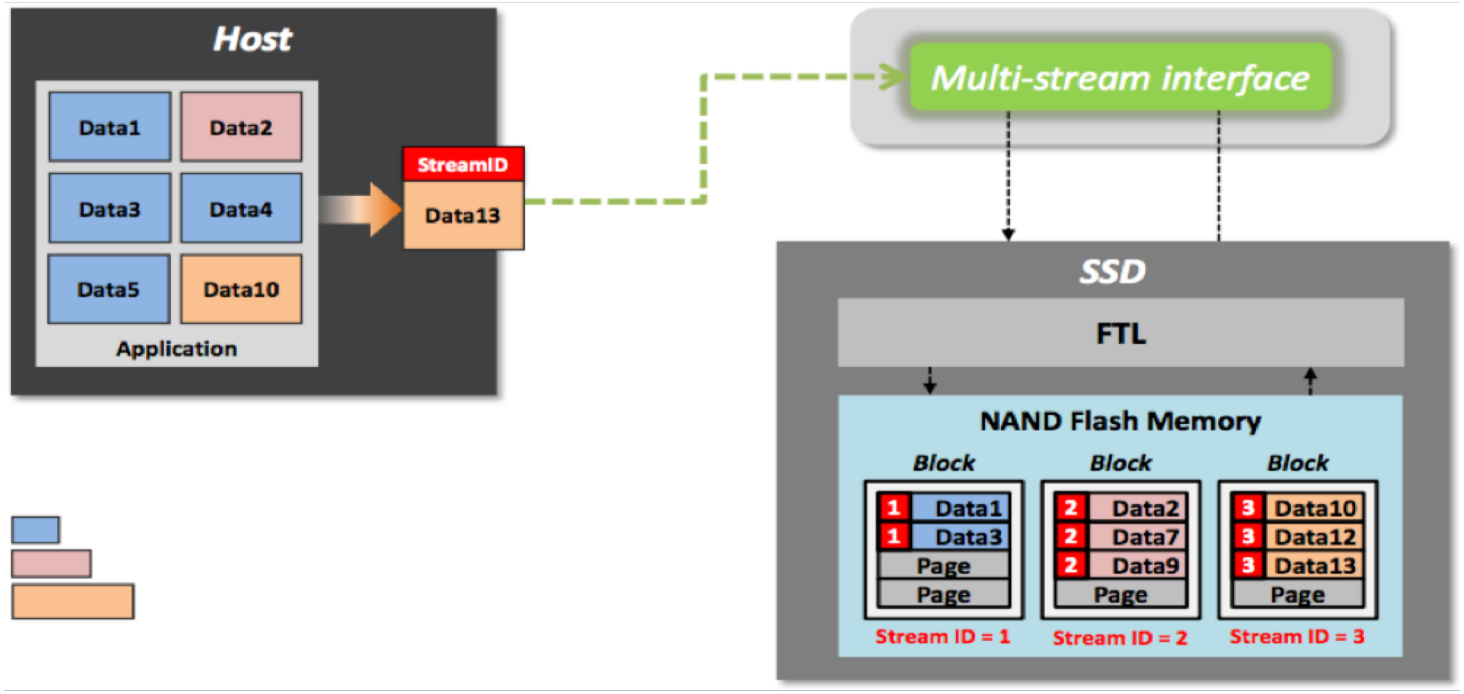
- **Write Amplification in Deployed Burst Buffer**
  - Write Amplification Factors (WAFs) usually between 2 and 3 in the SSDs of Cori's Burst Buffer



Reference) Glenn K. Lockwood, Burst Buffer NVMe Endurance Analysis

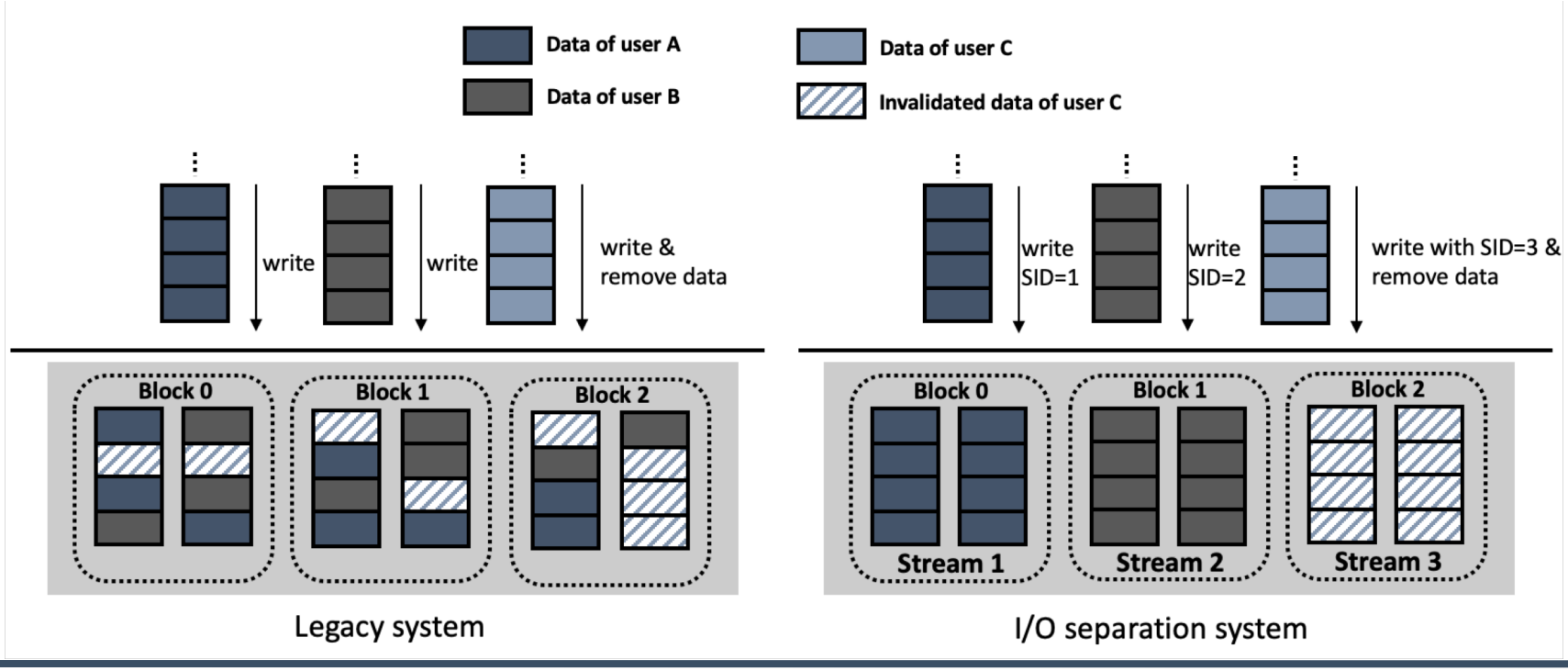
Multi-streamed SSD

- **Key idea of Multi-streamed SSD**
  - Mapping data with similar lifetimes to same stream
    - Increasing the probability that data within a block will be removed together
    - Reducing the number of copy operations during GC operation



I/O Separation Scheme in Burst Buffer

- **Data belonging to the same user should tend to have similar lifetimes in burst buffer environments**
- **Isolating user I/Os in separate SSD blocks using a multi-streamed SSD**
  - 4 copy and 1 erase vs just 1 erase operation

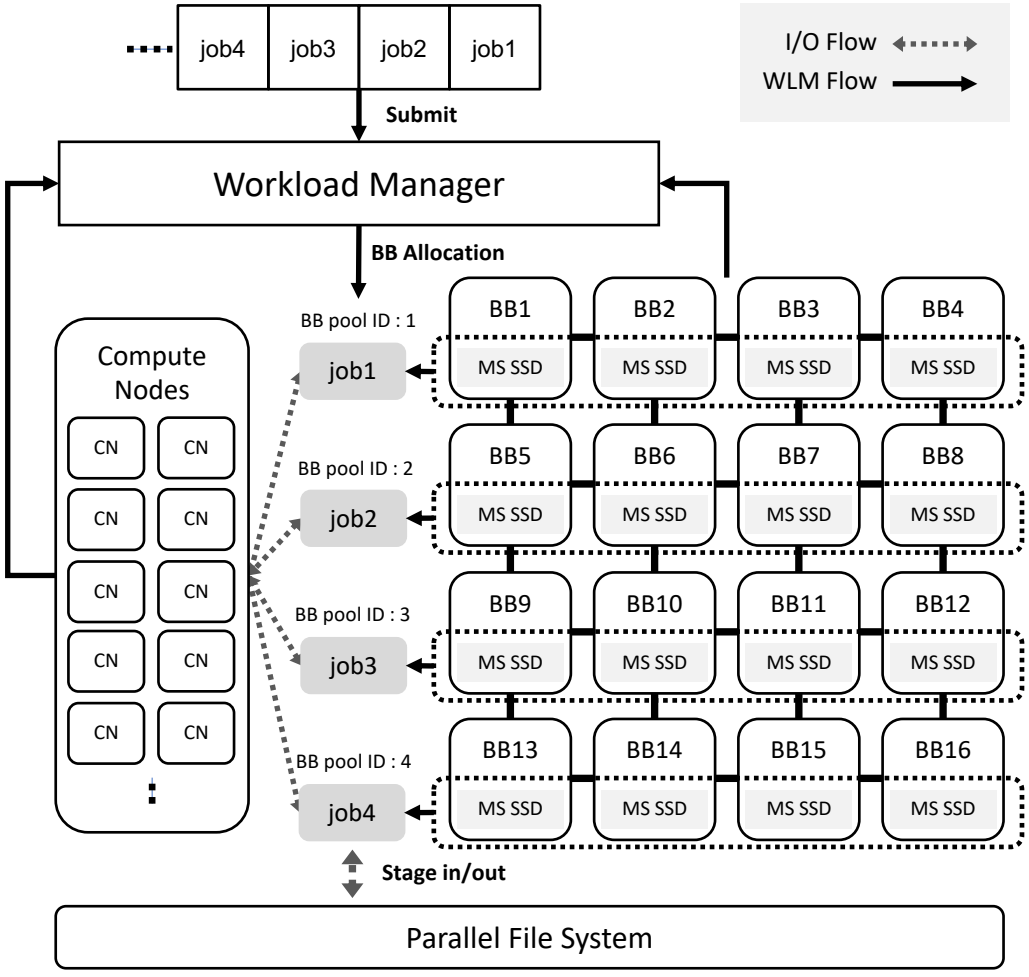


Challenges of Multi-stream Feature in BB

- **Limited Number of supported streams**
  - Multi-streamed SSD supports number of 4 to 16 streams
- **Striping I/O**
  - Reduction of available stream
- **User ID-based Stream Allocation**
  - Problem of skewed stream allocation

Framework for I/O separation scheme

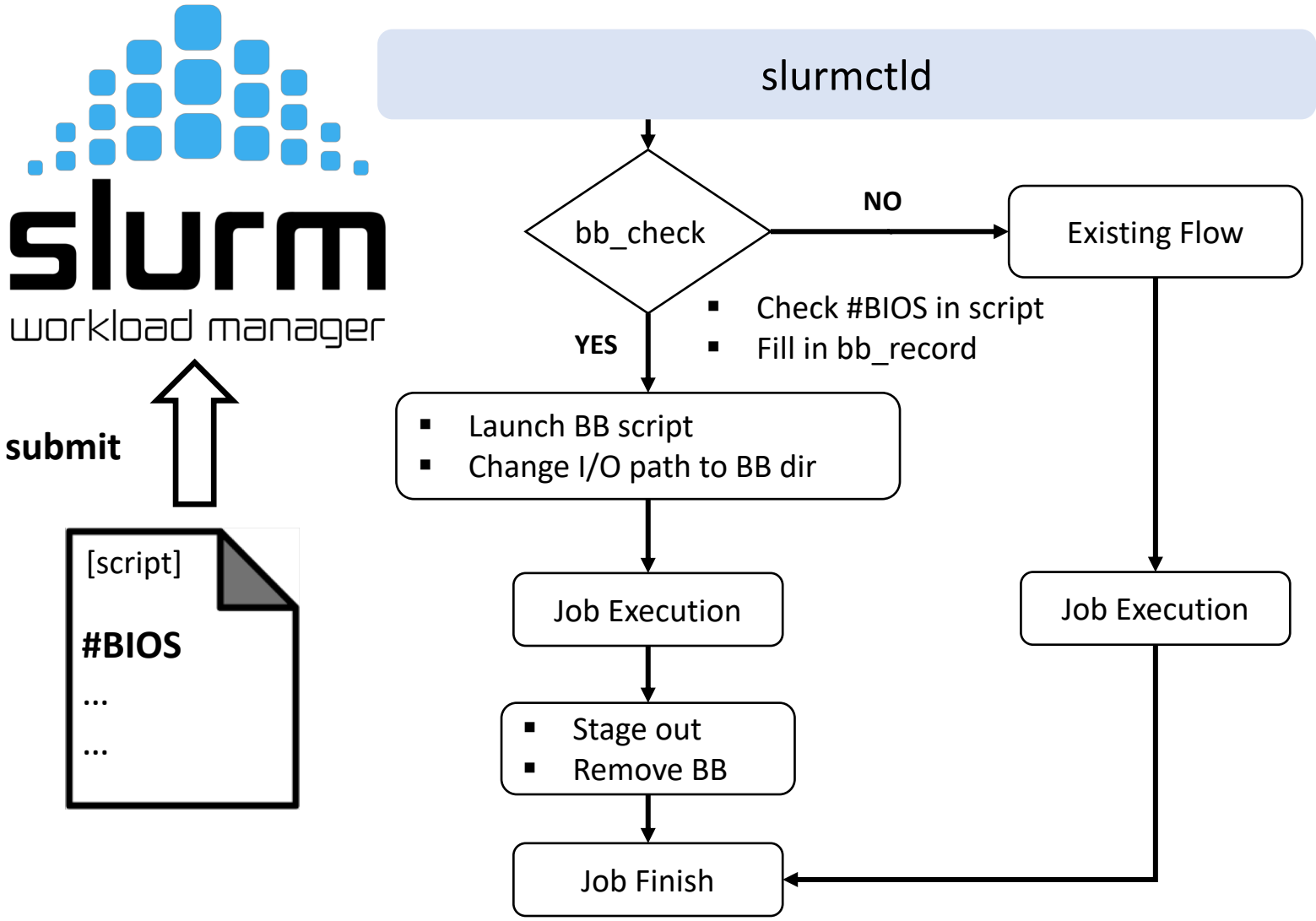
- **Integrating with workload manager**
  - Treat burst buffer as high-speed storage resource
- **Burst buffer pools**
  - BB allocation unit
- **Stream-aware scheduling policy**
  - Load balancing
  - User-ID based stream allocation



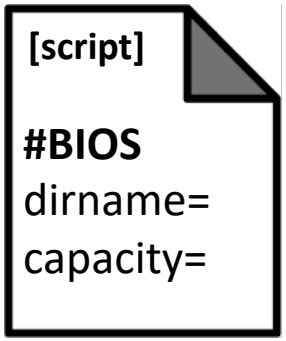
Directive for integrating with SLURM

- **#BIOS directive**
  - #BIOS dirname="dir_name"
    - Create $BB_HOME/user_name/dir_name
    - If user skip the "dirname", work as a existing flow
  - Capacity="size"
    - Assign the stripe count according to request size
    - BB granularity : 20GB
    - 0~20GB : 1, 20~80GB : 2 , 80GB or more : 4
- **Add logic in slurmctld code & script for setting BB**
  - Logic, SET_IOSBB.sh, EXIT_IOSBB.sh
    - SET_IOSBB.sh : creating BB space for job
    - EXIT_IOSBB.sh : stage-out & remove BB space

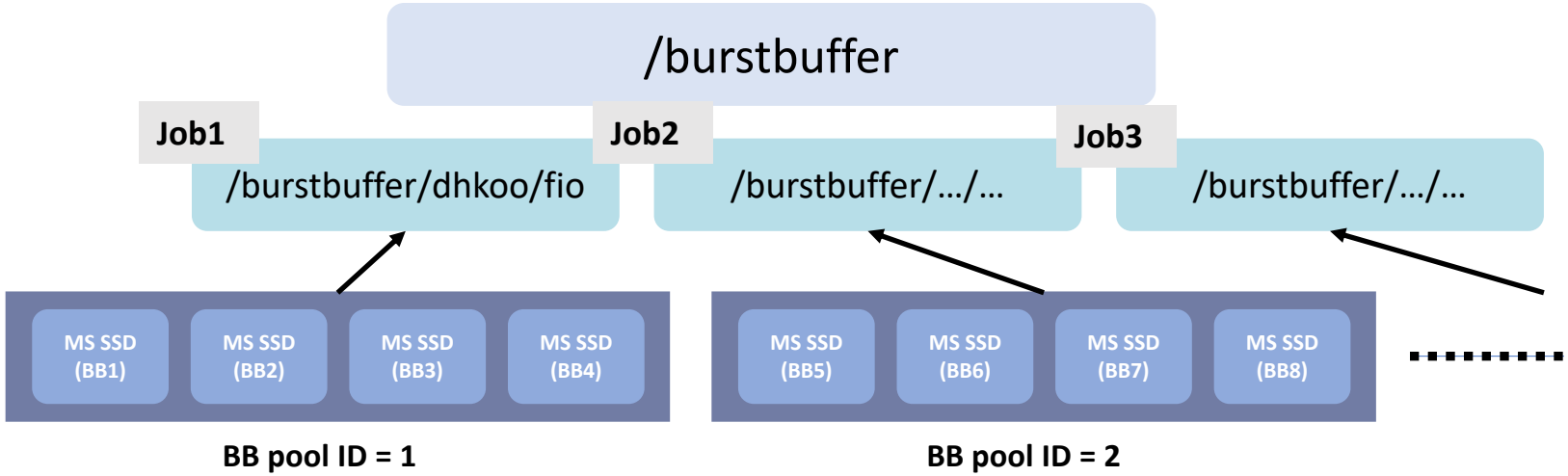
Logic for integrating with SLURM



Workflow in Framework



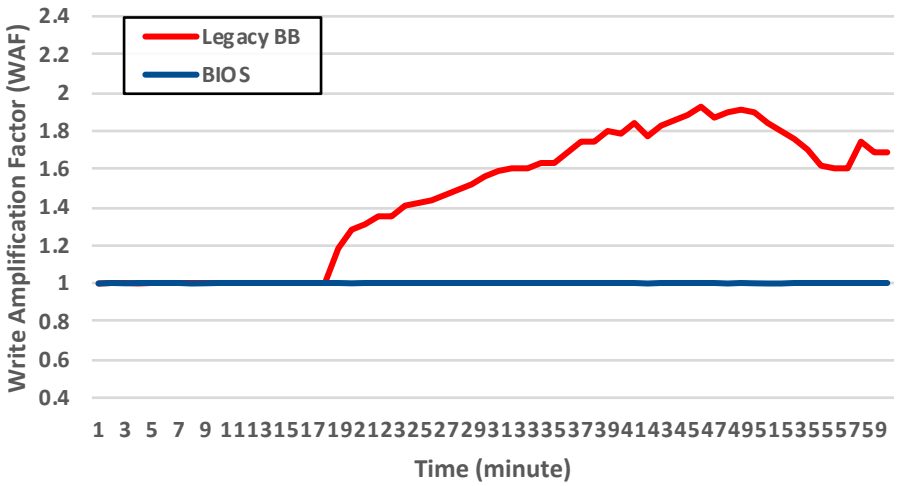
- **Create a dedicated directory for a job in BB**
  - /burstbuffer/{user_name}/{dirname}
- **Setting BB pool ID and stripe count for directory**



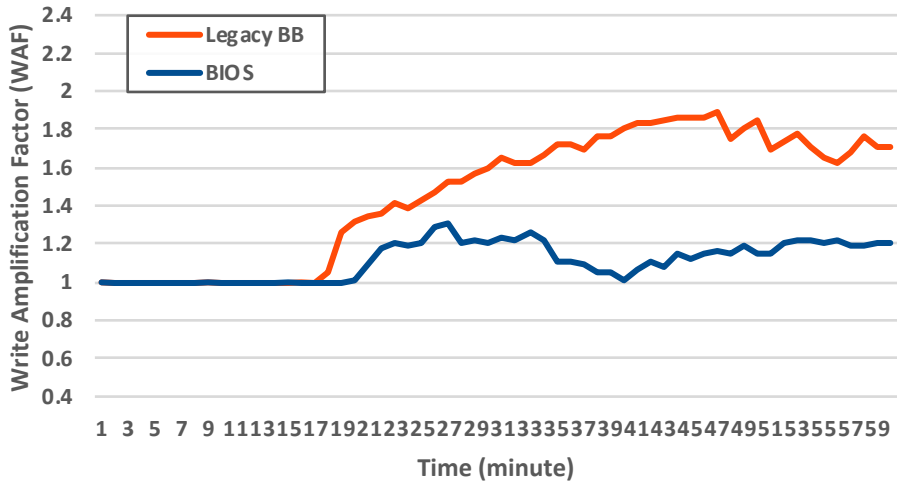
- **Assign idle stream-ID to a job based on user-ID**
- **Return stream-ID when it not used for a period of time**

Evaluation

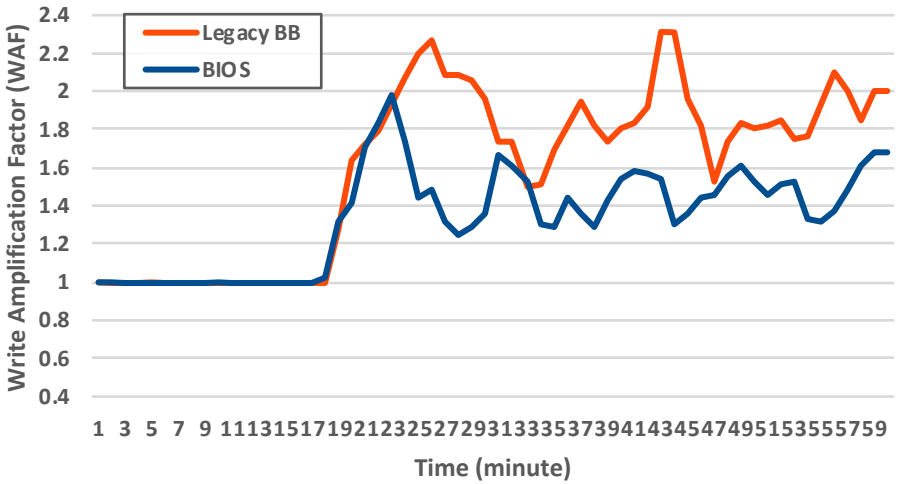
- **Legacy BB**
  - Increased up to 2 or more
- **BIOS (our proposed)**
  - Low and stable WAF
  - WAF is increased in conf 3, its average WAF is lower than all case of Legacy BB



Configuration 1



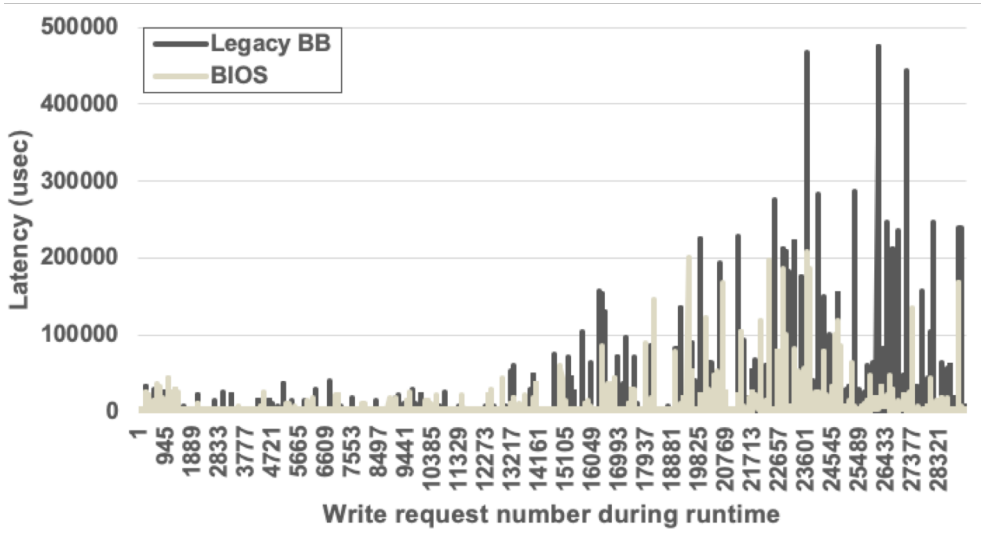
Configuration 2



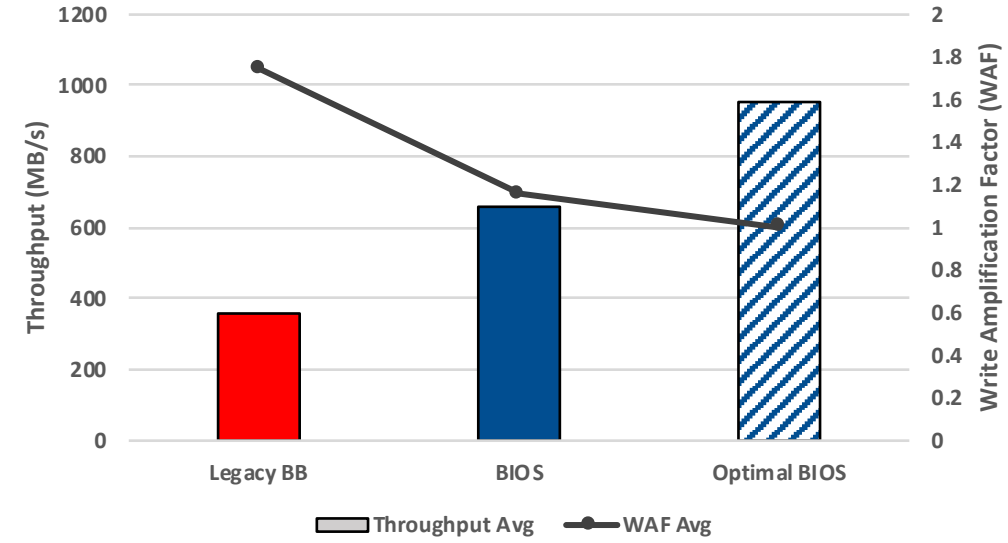
Configuration 3

Evaluation (Cont'd)

- **Comparison write latency between the BIOS and Legacy BB**
  - 1.72×, 1.93× and 2.04× faster at 90th, 95th and 99th



- **Results of average throughput and WAF on Legacy BB, BIOS and optimal BIOS**
  - WAF and Throughput Improved by 1.51x, 1.83x on BIOS and 1.74x, 2.66x on optimal BIOS



Conclusion

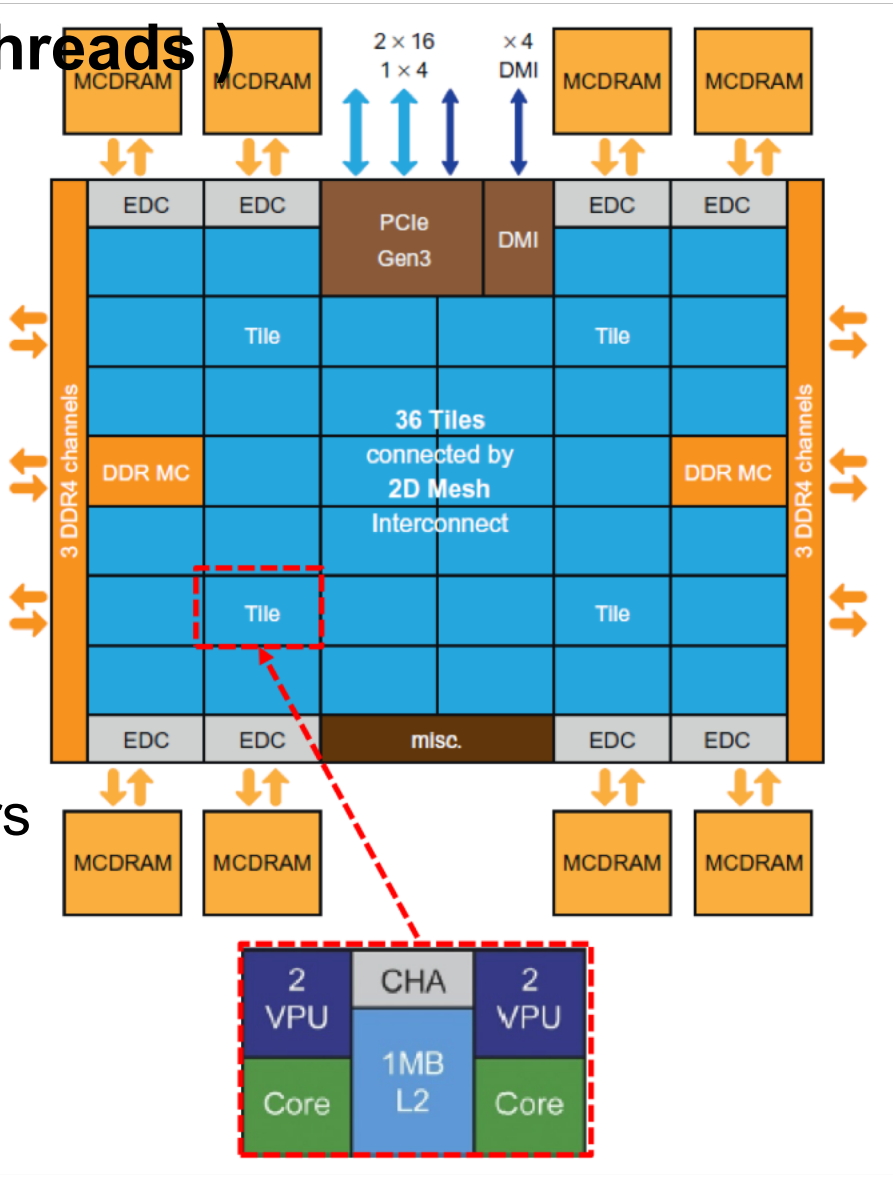
- **Bursty I/Os in a burst buffer cause write amplification in SSDs**
- **The benefits of multi-stream feature in BB can be reduced due to limited # of streams and striping I/O**
  - We implement the real burst buffer supporting multi-stream feature transparently
    - improve 1.20x endurance and 1.44x throughput on average
  - We implement the Framework for BIOS by integrating with workload manager with stream-aware scheduling policy
    - prove that it keeps on benefits of I/O separation scheme in burst buffer

고집적 매니코어 시스템 I/O 성능 특성 분석 및 최적 스토리지 시스템 연구

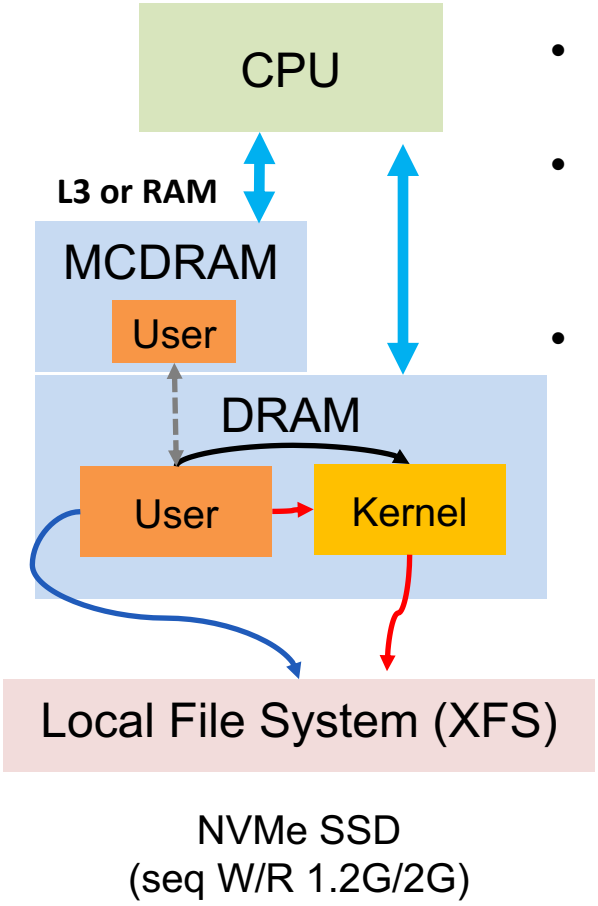
Research on analysis of I/O performance in highly integrated many-core system, and optimized storage system

Knights Landing (KNL) Architecture

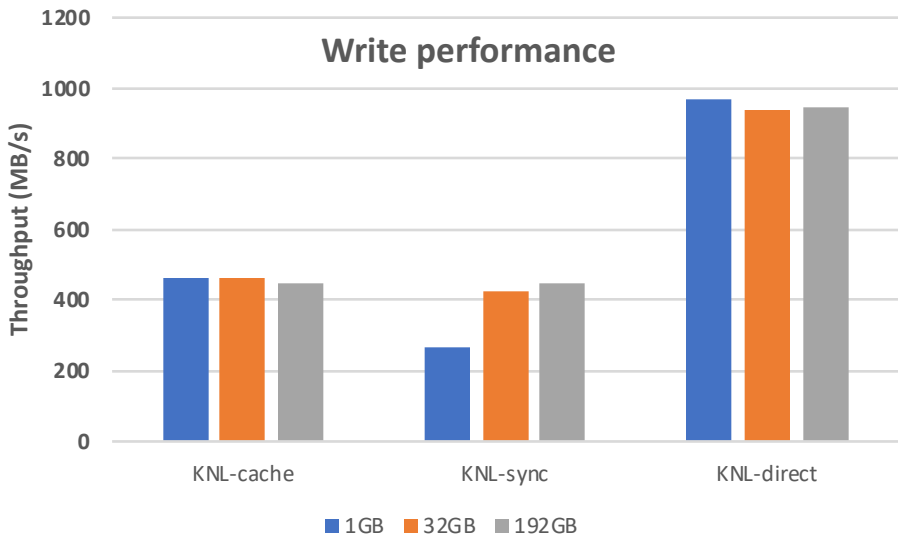
- **최대 36 tile (72 cores / 272 threads)**
  - 2 cores / tile
  - 1MB shared L2 cache
  - 2 * 512-bit VPUs/cores
- **2D mesh interconnect**
- **2 DDR memory controller**
- **6 channels DDR4**
  - Up to 90 GB/s
- **16 GB MCDRAM**
  - 8 embedded DRAM controllers
  - Up to 450 GB/s



Understanding the I/O performance on KNL node (1)

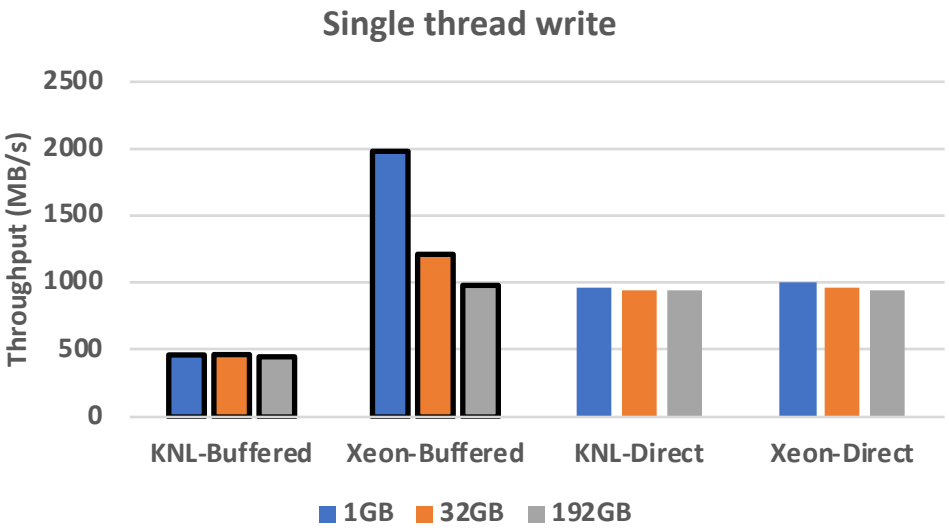


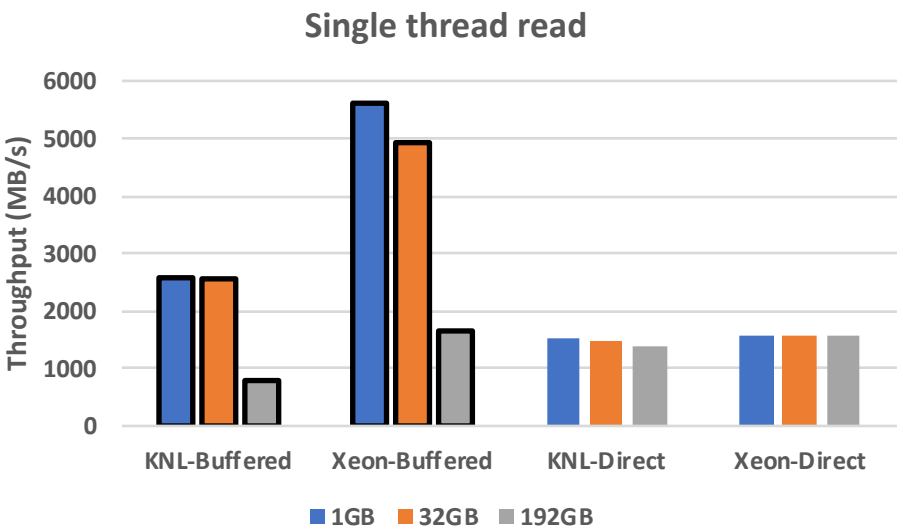
- **Buffered IO** : memcpy from user space to kernel space
- **Synchronous IO** : memcpy from user space to kernel space, and flush from kernel space to disk
- **Direct IO** : flush from user space to disk



Understanding the I/O performance on KNL node (1)

- **KNL vs Xeon (Buffered I/O and Direct I/O)**



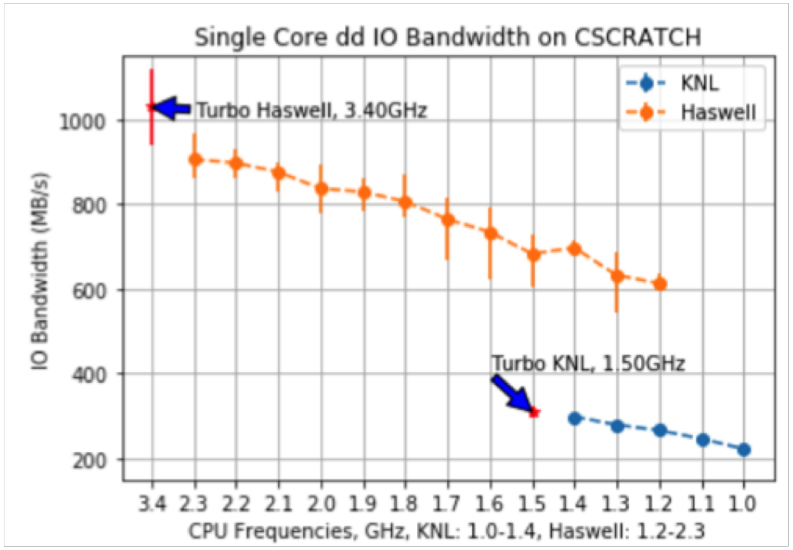


- **Performance Ratio KNL / Xeon**
  - Buffered write/read : 32.9% / 48.6%
  - Direct write/read : 97% / 93.7%

Understanding the I/O performance on KNL node (1)

- **Single thread execution time in I/O path**

<nano second>	KNL	Xeon	KNL/Xeon
write_begin	4884.28	990.93	4.93
flush_dcache_page	45.45	12.61	3.61
pagefault_disable	45.92	14.40	3.19
iov_iter_copy_from_user	1063.15	690.48	1.54
pagefault_enable	46.90	31.26	1.50
flush_dcache_page	43.78	10.72	4.09
mark_page_accessed	64.89	30.22	2.15
write_end	1040.68	273.89	3.80
cond_resched	59.00	15.33	3.85
iov_iter_advance	61.84	17.23	3.59
balance_dirty_pages_ratelimited	87.28	18.22	4.79
Total time	7443.18	2105.30	3.54



Buffered IO performance
with Varying CPU Frequencies, dd
(*Reference)

 # KNL
- Instructions : 2,721,005,578
- Insns per cycle : 0.49
- Write : 543MB/s

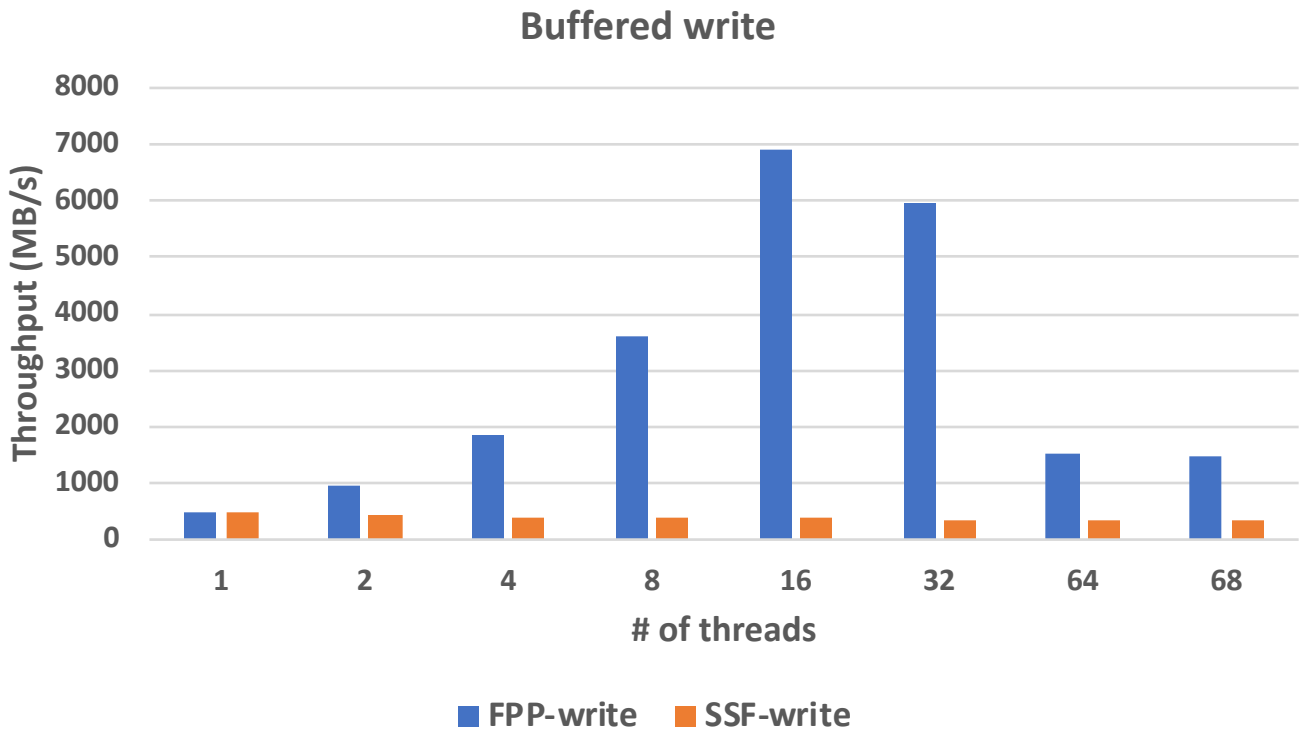
 # Xeon
- Instructions : 1,720,778,459
- Insns per cycle : 1.09
- Write : 1957MB/s

Low Frequency
Differences in CPU architecture

**Reference : Understanding the IO Performance Gap Between Cori KNL and Haswell (CUG17)*

Understanding the I/O performance on KNL node (2)

- **I/O Scalability on KNL**
  - Buffered I/O (File-per-process & Single-shared file)
  - # of threads : 1,2,4,8,16,32,64,68
  - File size per thread : 1GB



Understanding the I/O performance on KNL node (2)

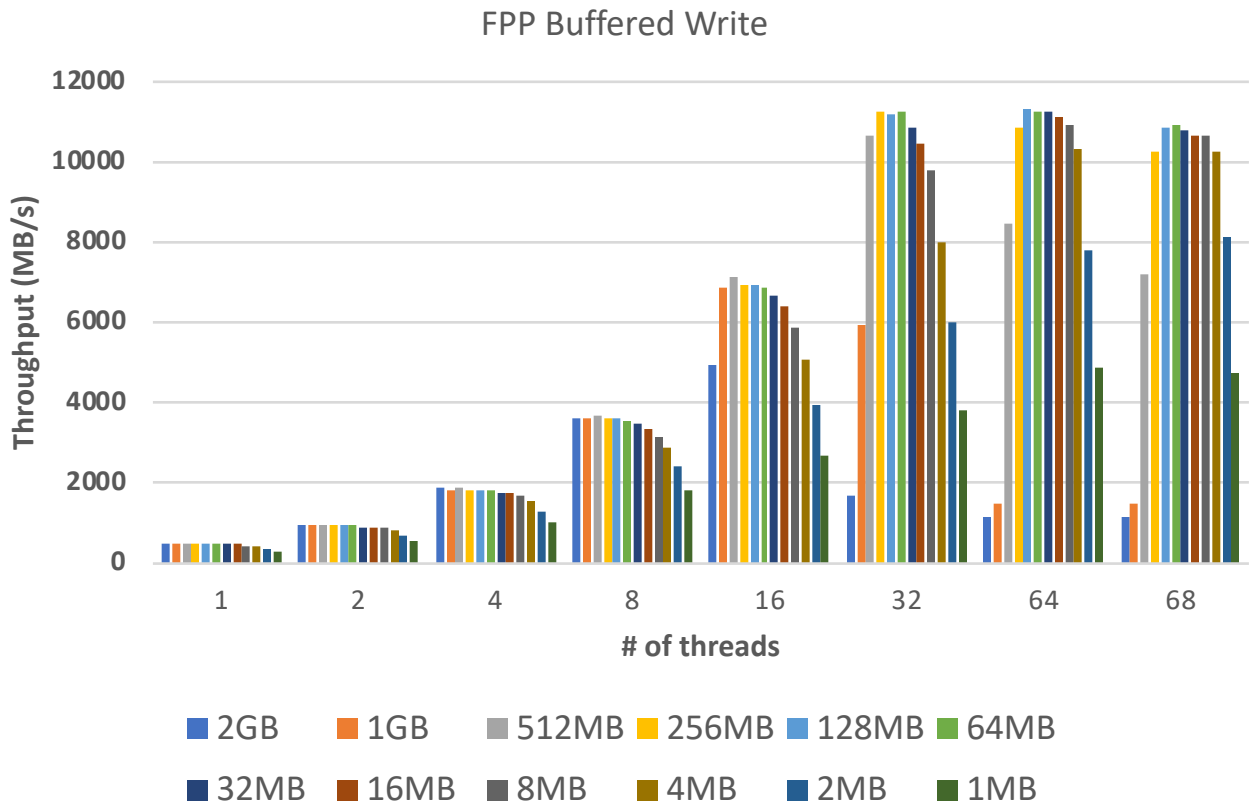
- **Average time per access**
  - X-axis : # of threads
  - File size per thread : 1GB

Average Time (ns)	1	2	4	8	16	32	64
write_begin	4884.3	4821.4	5045.7	5226.8	5582.4	6397.8	11105.5
flush_dcache_page	45.5	46.0	87.3	112.4	113.6	115.0	93.0
pagefault_disable	45.9	44.5	104.7	123.2	121.1	126.4	97.9
iov_iter_copy_from_user	1063.2	984.1	1159.5	1231.4	1207.8	1256.8	1163.7
pagefault_enable	46.9	47.7	101.4	125.2	125.0	124.0	96.4
flush_dcache_page	43.8	42.0	100.7	121.9	115.5	123.2	96.0
mark_page_accessed	64.9	67.1	131.5	146.6	145.3	148.7	117.4
write_end	1040.7	1041.4	1086.1	1103.4	1153.7	1215.7	1259.6
cond_resched	59.0	56.5	107.4	122.3	153.4	150.9	111.3
iov_iter_advance	61.8	62.4	121.5	139.0	132.8	135.8	111.1
balance_dirty_pages_ratelimited	87.3	88.9	152.7	164.7	160.8	7237.9	141488.2

- Kernel parameters associated with dirty pages
- vm.dirty_ratio : 40% (38.4GB)

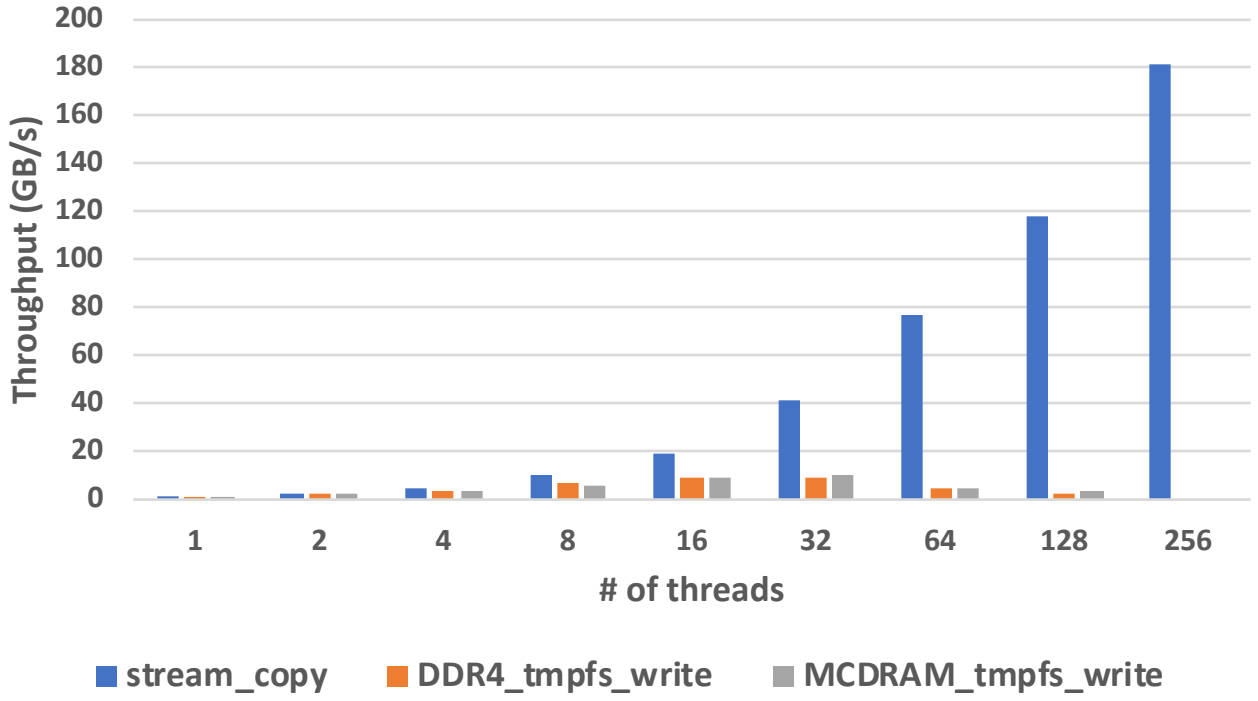
Understanding the I/O performance on KNL node (2)

- **Scalability of Write on KNL**
  - Buffered write (File-per-process)
  - Variables : # of threads, file size per each thread



Understanding the I/O performance on KNL node (3)

- **Does MCDARM provide better I/O performance?**
  - Stream benchmark, tmpfs(DDR4), tmpfs(MCDRAM)
  - Show a similar performance as the system memory
  - It's used effectively in memory operations



Conclusion

- **KNL shows poor performance in single buffered write**
  - Low frequency, difference in architecture
- **When application that uses the N-1 pattern's checkpointing is performed in KNL node, above phenomenon can be problem**
  - We propose the changing the N-1 pattern to N-N pattern using PLFS (Parallel Log-Structured File System)
- **In scalability evaluation, we revealed that cause of performance degradation in buffered write is dirty page management of kernel**
- **No benefits of MCDARM as a temporary storage**
  - I/O operation could not utilize the high bandwidth due to context switiching

클라우드 환경에서 이종저장장치를 활용한 하이브리드 적응형 스토리지 시스템

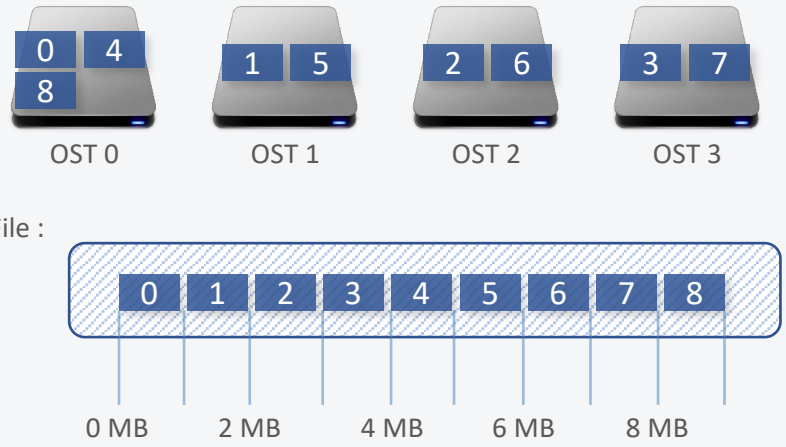
Adaptive hybrid storage systems leveraging heterogeneous storage devices in HPC cloud environments

Motivation and Proposal

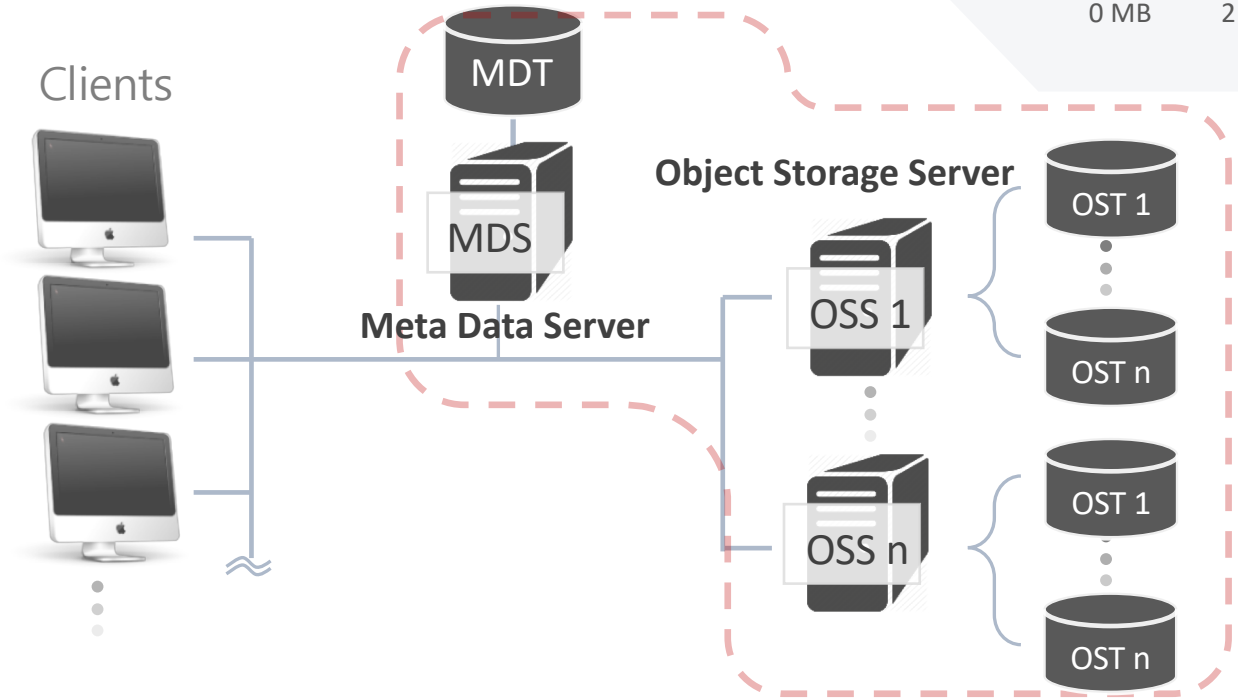
- **I/O patterns**
  - Advent of data-intensive workloads
  - Need for supporting various I/O patterns
- **Hard Disk Drive**
  - Mainly HDD devices have been used in the existing HPC environments
  - Commercialization of Solid-State-Disk
- **Utilizing Progressive File Layout leveraging SSDs for mixed sizes I/Os**
  - Propose an optimized Progressive File Layout (PFL) scheme which can effectively leverage heterogeneous storage devices in Lustre file system

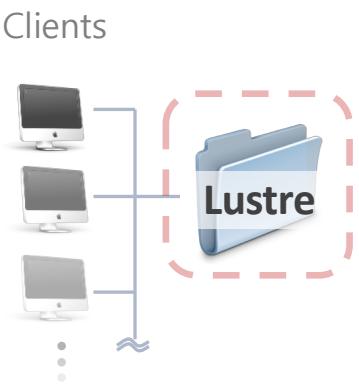
Lustre File System

File System → Distributed File System → Lustre File System



Architecture





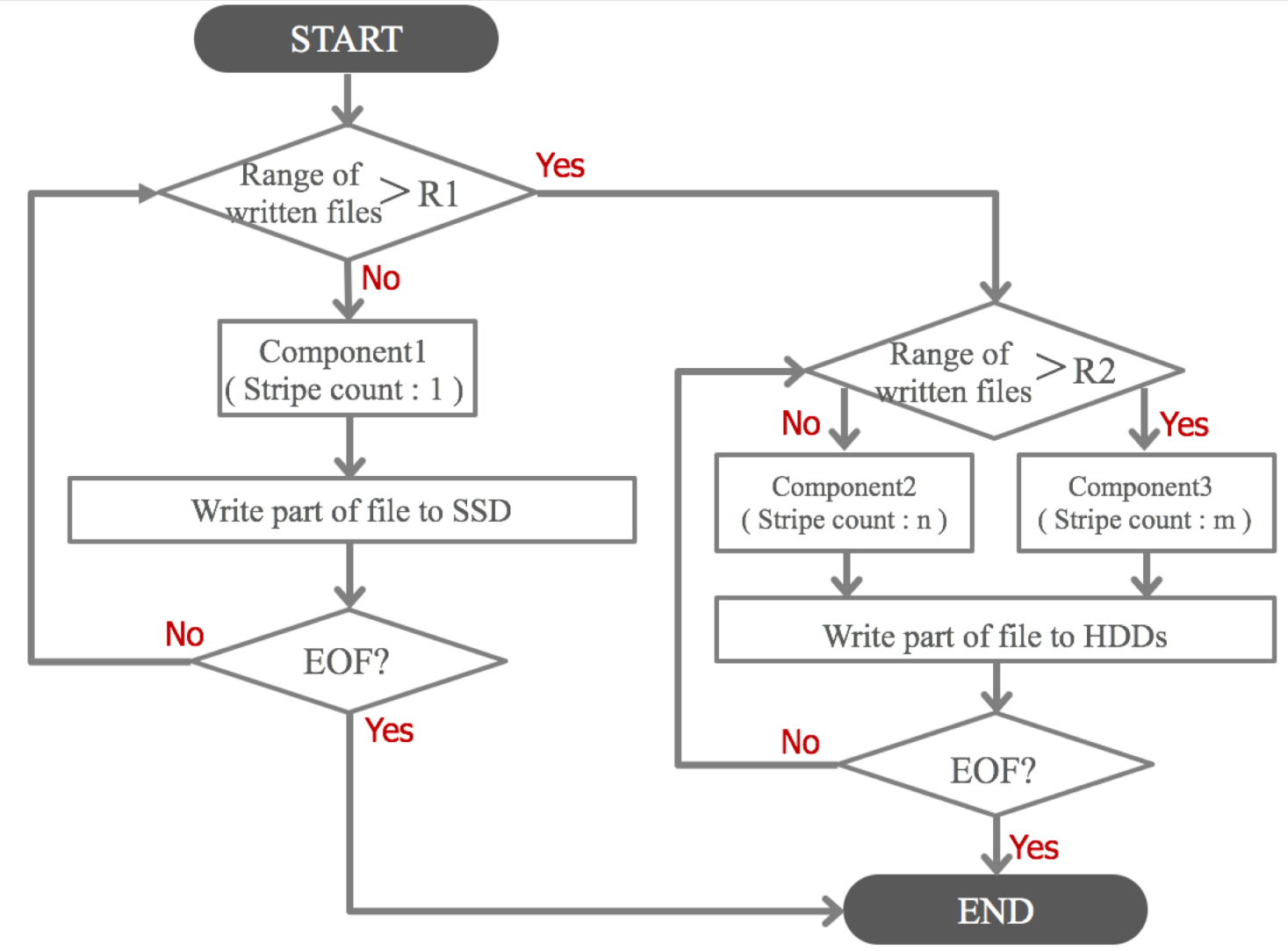
Progressive File Layout

[0 , ⓐ) of file is stored in 1 OST (1 stripe)

[ⓐ,ⓑ) of file is stored in 4 OSTs (4 stripe)

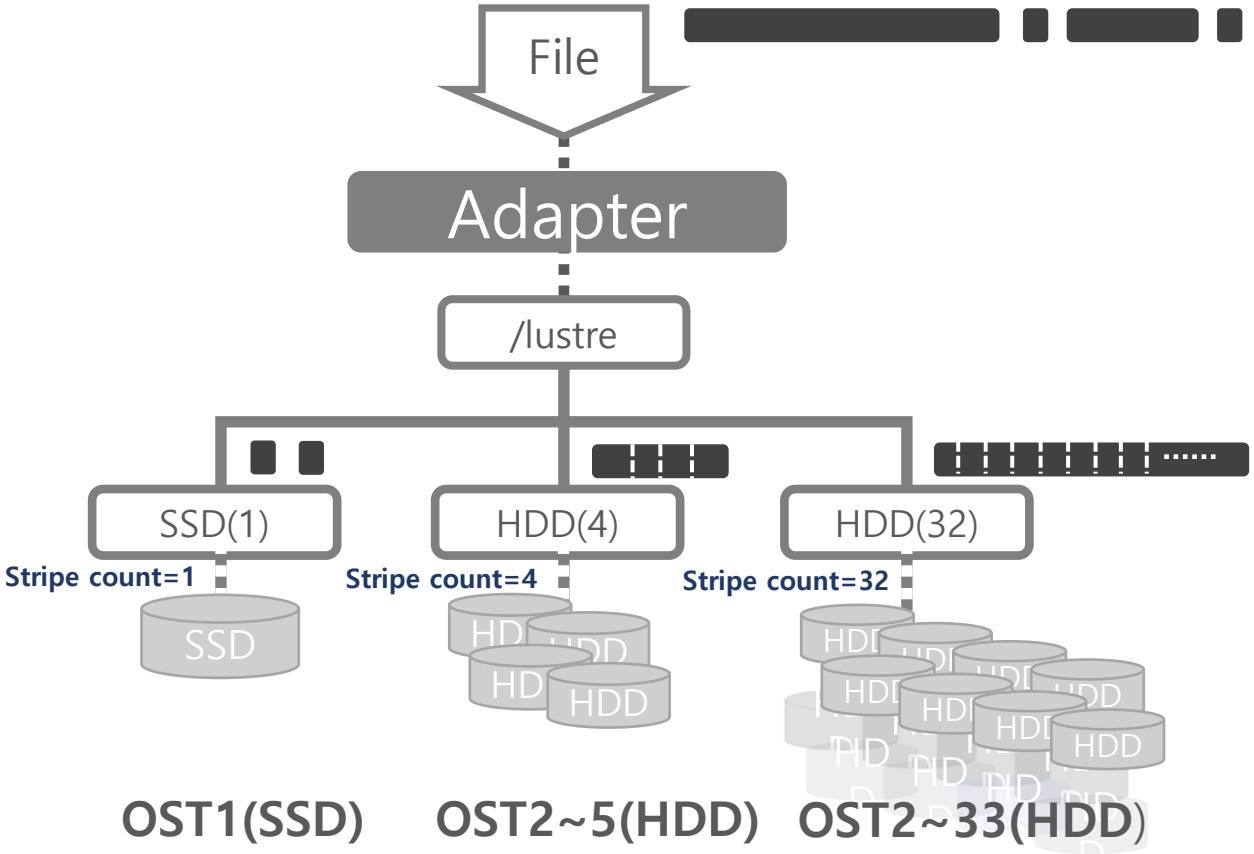
[ⓑ,ⓒ) of file is stored in 32 OSTs (32 stripe)

Flowchart of PFL in Lustre

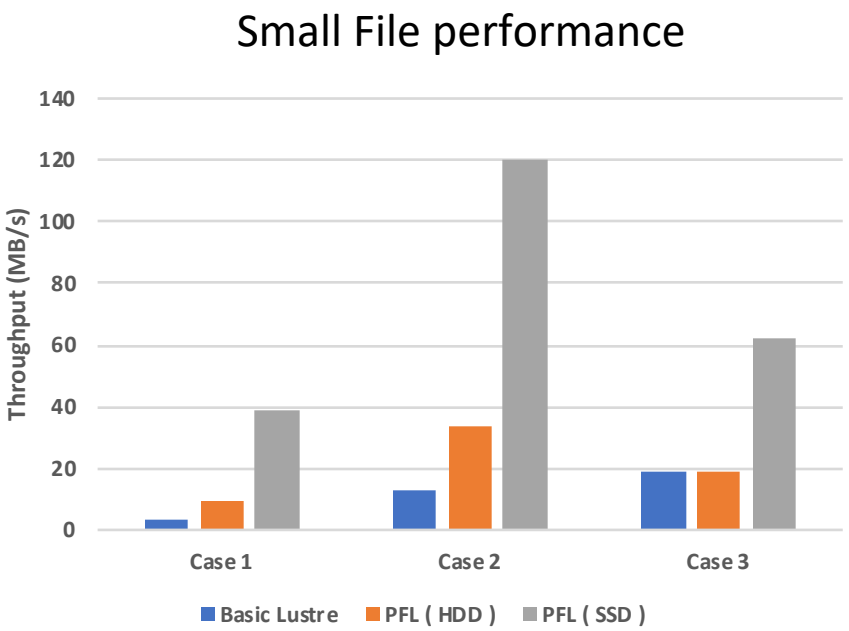


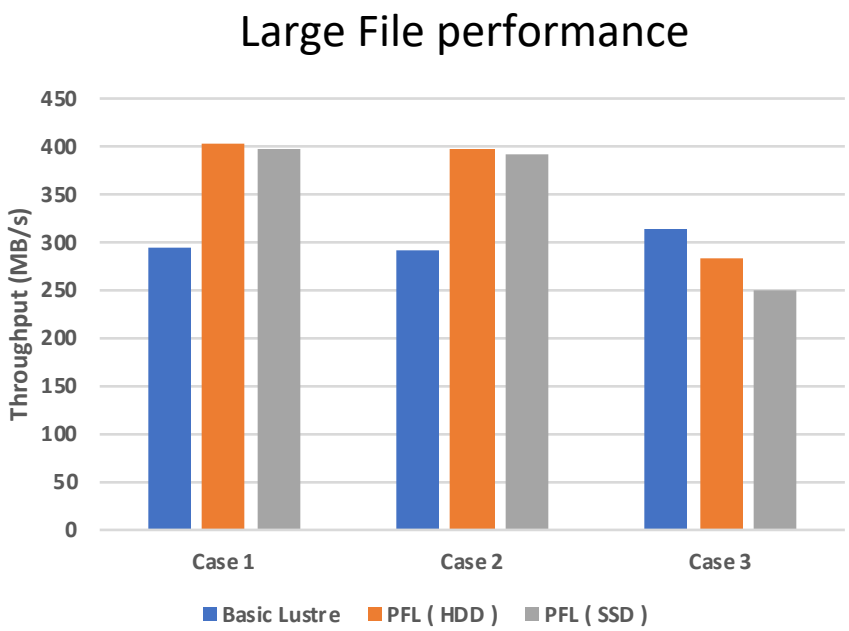
System using progressive file layout with SSD

- **When the various size of files are incoming to Lustre**
  - Adapter passes the file to a dedicated directory with different striping configuration according to file size



Evaluation with extensive experiments





- **Experimental targets**
  - Basic Lustre, PFL(HDD) and PFL(SSD)
- **Experimental configurations**
  - Case1,2 and 3 consist of 1,4 and 8MB of small files respectively with 16GB large file

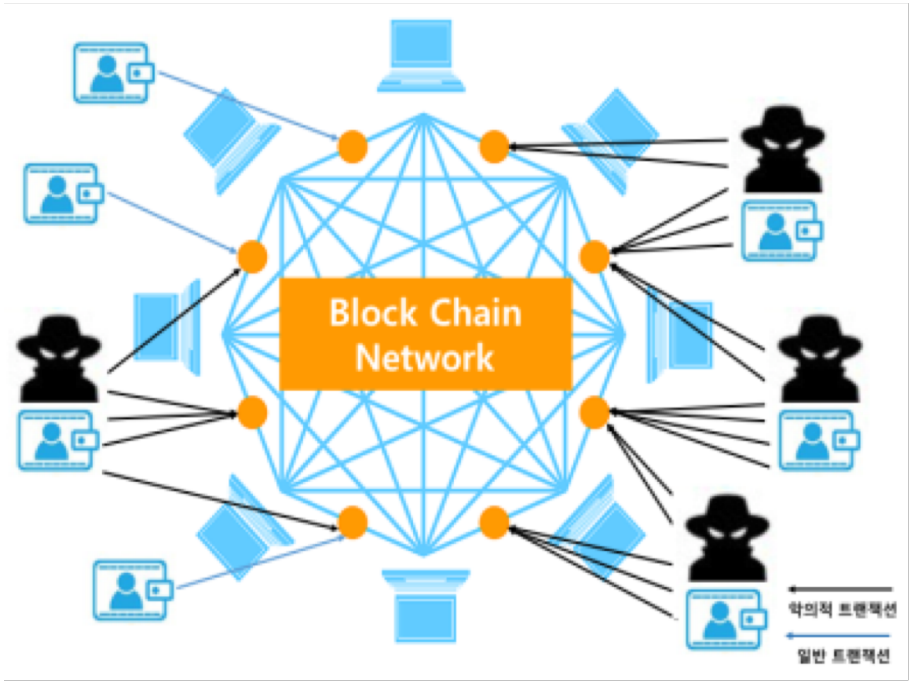
Conclusion

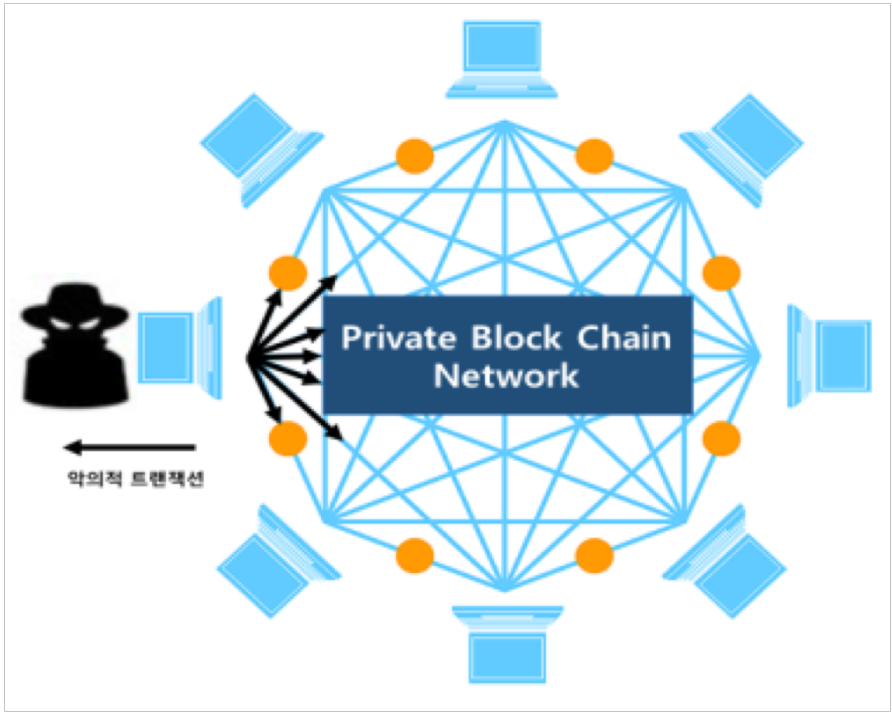
- Combining SSDs with the PFL can maximize small I/O performance while preserving competitive large file I/O throughput
- The reference points which is criterion for distinguishing files in the PFL will optimize the total system I/O performance
- With a large number of small I/O threads, which occurs in typical cloud environments, the small I/O throughput of the PFL(SSD) is shown to be up to 6.5 times as high as those of the PFL(HDD) and the legacy Lustre

ETC

Research outline

- 프라이빗 블록체인 기반 범용 데이터베이스 플랫폼의 보안 감지 및 대응을 위한 시스템 필요
- 프라이빗 블록체인에서 발생할 수 있는 공격 감지
  - 블록체인 DDoS 공격 감지
  - 노드별 트랜잭션 검증





Research Activities

- **블록체인 네트워크 모니터링**
  - 이더리움 하모니를 활용한 블록체인 네트워크 모니터링
  - 기존 하모니의 불필요 요소 삭제 작업 진행
  - 블록정보 및 트랜잭션 정보 나열 메뉴 페이지 추가

- **위험성 감지 및 알람 기능**
  - Go 이더리움에서 유효하지 않은 트랜잭션에 대한 로그 정보를 기록하는 함수 추가
  - 로그 정보 분석 및 메일링 기능 담당 데몬 구현
  - 유효하지 않은 트랜잭션에 대한 주기적 감시 및 알림 기능 제공
  - DDoS 공격에 대한 위험성 감지 기능 제공

Research results

- 블록체인 네트워크 전체적인 모니터링 기능 제공
- 악의적인 트랜잭션 및 DDoS 공격 감지 및 알림 기능 제공