KVSSD 신규 프로젝트 개발 기획서

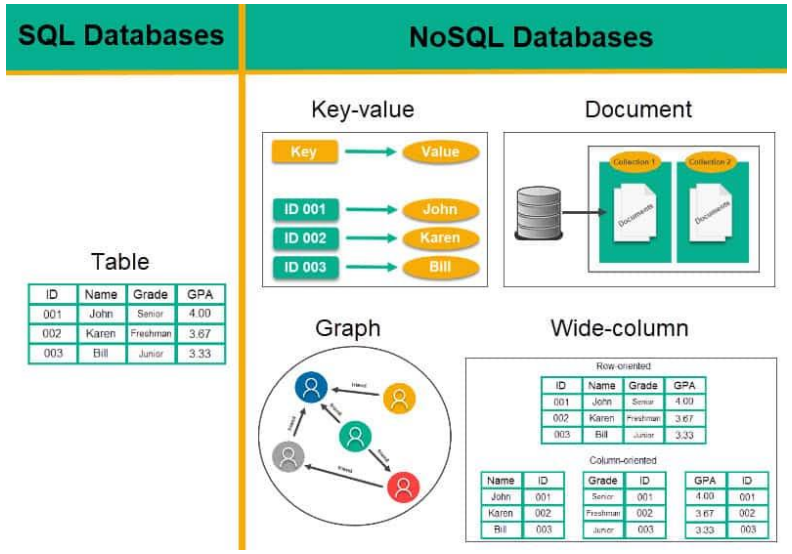
오기준

목차

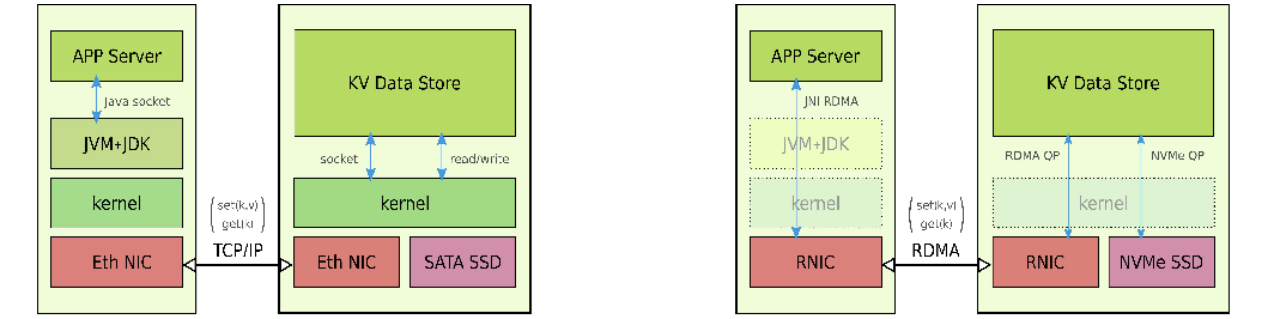
개요 및 목표

개요

- 키-밸류 데이터베이스는 기존 관계형 데이터베이스의 한계를 벗어나서 확장성, 사용 편의성, 성능상의 이점을 얻기 위해 생긴 새로운 방식이다.
- 기존 SSD 인터페이스는 관계형 데이터베이스의 블록 기반 쓰기/읽기에 중점이 되었다. 이를 위해 키-밸류 데이터베이스들은 별도 블록 생성 과정을 거치게 되면서, SSD 성능을 극대화해서 사용하지 못하는 문제가 존재했다. 이를 해결하기 위해 키-밸류 인터페이스를 지원하는 SSD에 대한 연구가 진행되었고, 다양한 이점으로 인해 NVMe 인터페이스 표준에도 들어가게 되었다[1].



출처: https://phoenixnap.com/kb/sql-vs-nosql



(a) Conventional I/O stack for key-value middleware (b) An *all-in-userland* approach for key-value middleware

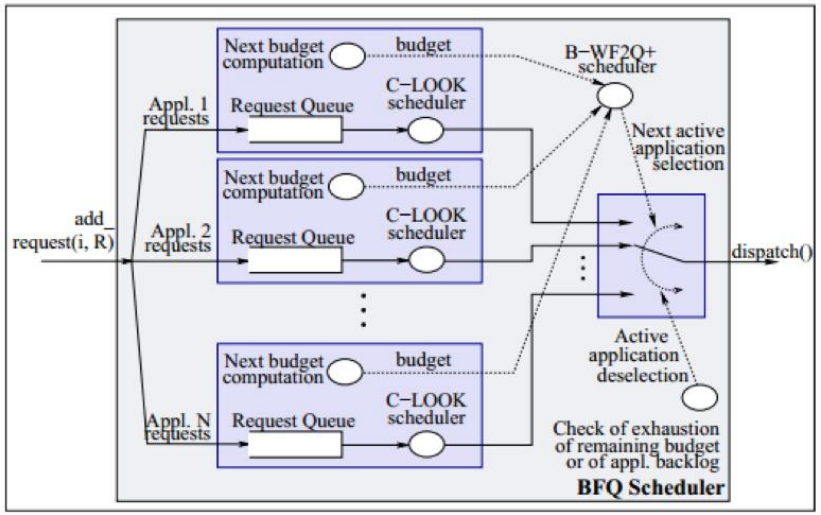
Figure 1. Target key-value middleware architecture. In our design, both the OS kernel and runtime JVM are bypassed.

출처: An, Z., Zhang, Z., Li, Q., Xing, J., Du, H., Wang, Z., Huo, Z., & Ma, J. (2017). Optimizing the Datapath for Key-value Middleware with NVMe SSDs over RDMA Interconnects. 2017 IEEE International Conference on Cluster Computing (CLUSTER), 582-586.

개요 및 목표 (Cont'd)

개요

- 그러나 최근 클라우드 컴퓨팅 시대의 도래로 단일 노드에는 하나의 서비스가 아니라 여러 서비스가 들어갈 수 있게 되면서, 단일 노드에 여러 사용자의 리소스 할당이 가능하게 되었고[2], 이로 인해 시끄러운 이웃(Noisy Neighbor) 문제가 발생하기 시작했다[3].
- I/O 과정에서의 해당 문제를 해결하기 위해 다양한 I/O 스케쥴링 방식을 커널 단에서 제공한다[4]. 하지만 대다수 블록 기반의 I/O만을 대응하고 있거나, SSD 내부 칩 구성에 상관없이 데이터가 쓰여진다. 이로 인해, 키-밸류 SSD의 성능을 온전히 다 사용하지 않는 문제를 가지고 있다.



출처: https://www.cnblogs.com/Linux-tech/p/12961283.html

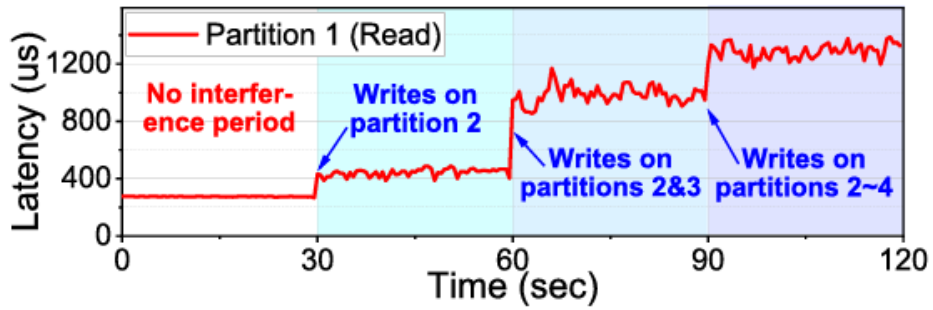


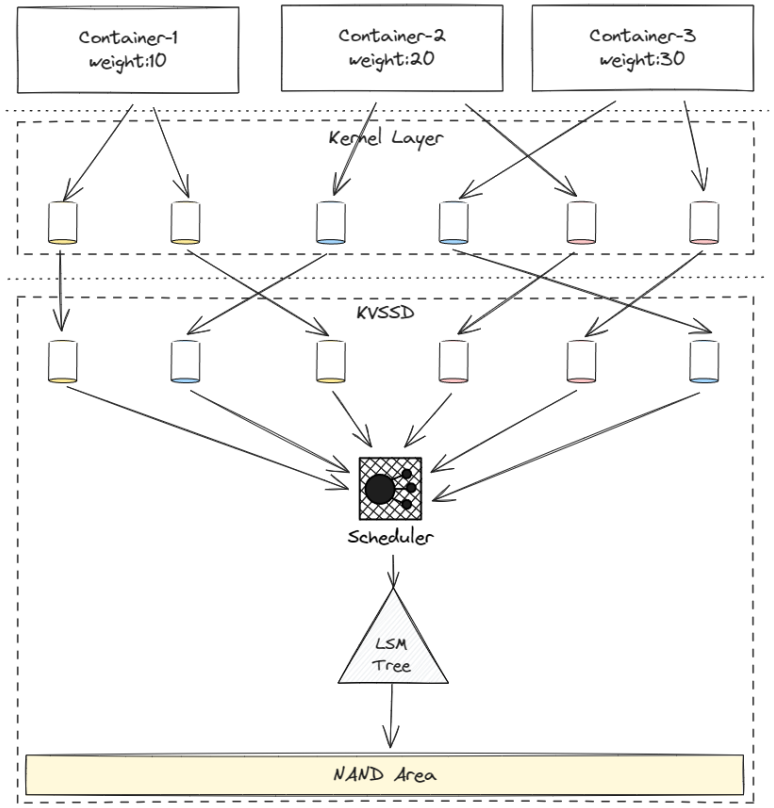
Figure 1: Interference in storage sharing.

출처: Kwon, M., Gouk, D., Lee, C., Kim, B., Hwang, J., & Jung, M. (2020). DC-Store: Eliminating noisy neighbor containers using deterministic I/O performance and resource isolation. In 18th USENIX Conference on File and Storage Technologies (FAST 20) (pp. 183-191).

개요 및 목표 (Cont'd)

목표

- 기존 커널 영역에 있던 I/O 스케쥴러를 디바이스 레벨로 내리고, SSD 내부 아키텍처를 고려한 배치를 수행하여 클라우드 환경에서의 키-밸류 SSD 성능을 극대화한다.



개발 계획

작업 리스트

대분류	소분류	난이도	작업 ID	작업	내용	비고
Preparation	Setting Up a Development Environment	Low	P1	리눅스 커널 및 NVMeVirt 개발 환경 구축	1. 커스텀 리눅스 커널 개발 방법 및 리눅스 디바이스 드라이버 개발 방법에 대한 이해 2. 커널 디버그 툴인 ftrace 또는 eBPF 사용법 숙지 3. 커널 개발 환경 구축 및 빌드 수행 4. 디바이스 드라이버 개발 환경 구축 및 빌드 수행 5. NVMeVirt 개발 환경 구축 및 빌드 수행	공통 과제 참고 자료 • 디버깅을 통해 배우는 리눅스 커널 • Linux Kernel Development (3rd ed.) 원서 • 서울대 故 고건 교수님 강의 • Linux Device Drivers (3rd ed.) • Guide to building the Linux Kernel
	Literature Review	Low	P2	선행 연구 분석	1. I/O Scheduler 관련 논문(BFQ, MQFQ, D2FQ): 기존 I/O Scheduler를 이해하고 Device-Level로 옮길 수 있는 대상을 선택 2. NVMeVirt 관련 논문: NVMeVirt의 동작 방식 및 추구하는 바가 무엇인지에 대한 이해 3. 기타 필요 논문	공통 과제 논문 추천 관련 이메일 참조

개발 계획 (Cont'd)

작업 리스트

대분류	소분류	난이도	작업 ID	작업	내용	비고
Full-Stack Development	Test Tool	Mid	FSD1	KVSSD에 표준 KV 커맨드 전송해서 받는 것을 점검	1. NVMeVirt에서 KVSSD로 구축 수행 2. xNVMe를 참고해서 KV 명령어 전송	xNVMe's KV impl.
Host-level Development	User Space	High	US1	YCSB-cpp 커스텀 데이터베이스 개발	1. Key/Value를 받아서 메모리에 저장하는 커스텀 데이터베이스 개발 2. 메모리에 저장하는 부분을 NVMe 표준 KV 커맨드로 변경하도록 개선	https://github.com/ls4154/YCSB-cpp
		Mid	US2	KVSSD 정합성 확인 프로그램 개발	1. KVSSD가 주어진 명령에 따라 올바른 값을 반환하는지를 확인하는 프로그램 개발	https://github.com/BlaCkinkGJ/Flash-Board-Tester
	Kernel Space	Mid	KS1	I/O 스케쥴러 소스 코드 확인 및 실행 테스트	1. 각 I/O 스케쥴러의 구현을 커널에서 분석 2. 각 I/O 스케쥴러를 직접 실행하고 평가를 진행 3. 가능하다면 NVMe KV 커맨드를 날릴 때 I/O Scheduler를 활성화해서 같이 동작하는지 확인	
		Mid	KS2	Cgroup V1 동작 방식에 대한 이해	1. Cgroup V1을 실제로 사용해서 리소스 관리를 직접 수행 2. I/O 스케쥴러를 BFQ로 설정하고 Cgroup weight를 설정했을 때의 동작 과정을 분석 3. 커널 소스에서 cgroup관련 정보를 디바이스 드라이버까지 내릴 수 있는지를 확인	cgroup manual BFQ documentation

개발 계획 (Cont'd)

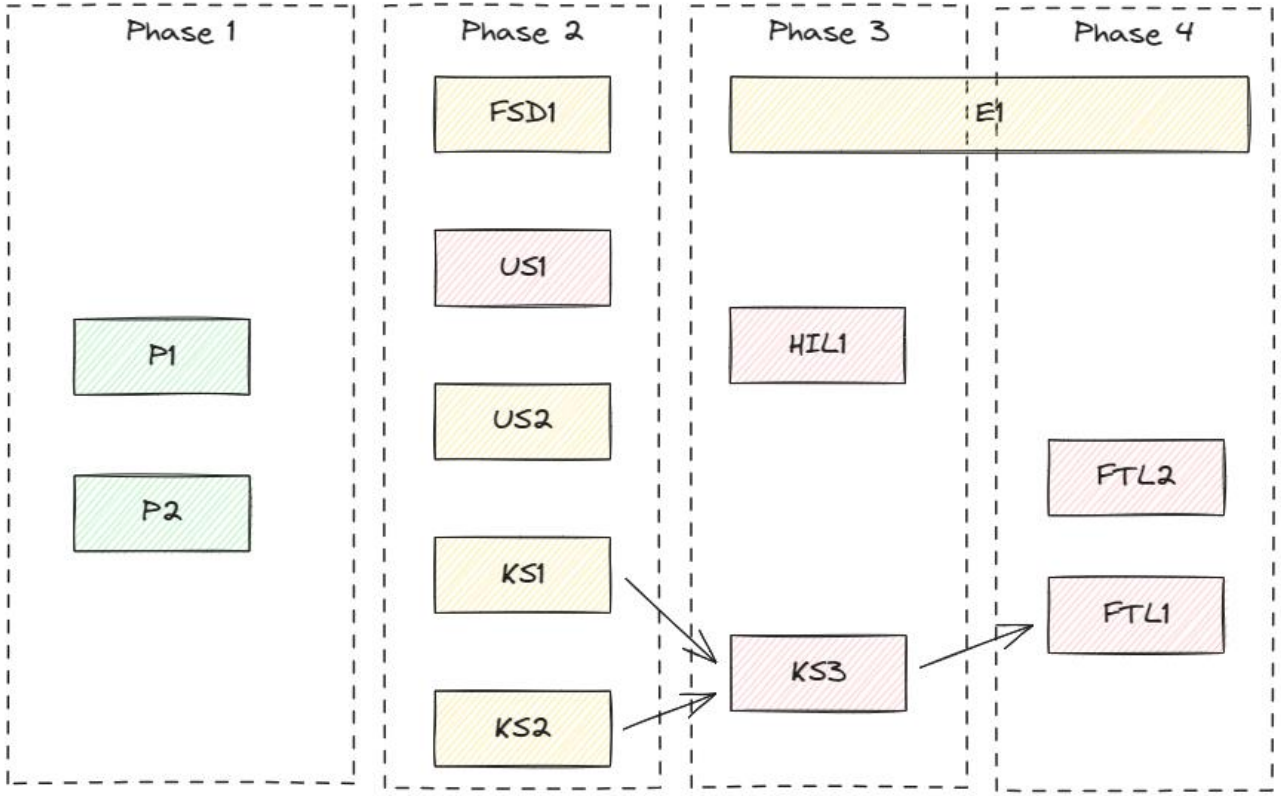
작업 리스트

대분류	소분류	난이도	작업 ID	작업	내용	비고
Host-level Development	Kernel Space	High	KS3	Cgroup weight가 NVMe 커맨드에 적재되도록 개선	1. Cgroup weight를 디바이스 드라이버 단에서 weight가 NVMe 커맨드에 적재될 수 있도록 개선	만약 cgroup을 통해 weight 전송이 힘든 경우 사용자 영역에서 NVMe reserved area를 이용해서 weight를 날리는 것으로 변경
Device-level Development	Hardware Interface Layer (HIL)	High	HIL1	Weight에 따라 I/O 커맨드 스케쥴링 수행	1. I/O Queue를 개발하고 weight 값에 따라 커맨드가 스케쥴링이 되도록 개발	
	Flash Translation Layer (FTL)	Mid	FTL1	I/O 스케쥴링된 커맨드를 LSM Tree에 삽입	1. HIL에서 받은 I/O를 LSM Tree에 삽입하고 적절한 값을 반환하도록 개발	HIL쪽 개발이 완료되어야 함
		High	FTL2	Weight에 따라 I/O를 분류	1. Weight에 따라 서로 다른 채널의 NAND에 배치되도록 LSM Tree를 개선	선택적으로 시간이 남으면 개발하거나 향후 과제로 남겨놓을 것
Evaluation		Mid	E1	성능 평가 진행	1. 앞서 제작한 YCSB 인스턴스를 여러 개 생성해서 weight를 달리하여 실행 2. 성능 평가를 어떻게 할 것인지를 결정	D2FQ 실험 내용 참조

개발 계획 (Cont'd)

작업 의존성

- 선은 직접적인 의존성을 나타냄
- 작업 ID는 이전 표 참조, 색은 난이도를 의미 (초록색: 쉬움, 노랑색: 보통, 빨간색: 어려움)



개발 계획 (Cont'd)

향후 확인 사항

- 각 태스크 진행 상황 트랙킹을 위한 Kanban 보드 구축 (GitHub project?)
  - Kanban 보드에서 각자 태스크 선택 후 작업 진행
- 개발 및 연구 과정 기록물 관리 방법에 대한 논의 (Google docs?)
- Kick-off 모임 주최 (개발 환경 구축 모임도 같이해서?)
- 작업별 repository 생성 후, 개발 시 유의 사항 전달(GitHub 사용법이나 Coding Conventions)
- 리눅스 커널 개발이나 관련된 내용에 대한 오프라인 세미나 개최
- 업무 보고 방식 및 주기 정의 (분기별 오프라인 보고 필요?)

계획 일정

3인 기준으로 작성

- 명당 1개 난이도 High 작업과 2개 난이도 Mid 작업을 처리하는 것을 권고
- 난이도 High 작업 이후에는 반드시 난이도 Mid 작업을 수행 (High → High 금지)

대분류	소분류	난이도	작업 ID	2024년						2025년			
				07월	08월	09월	10월	11월	12월	01월	02월	03월	04월
Preparation	Setting Up a Development Environment	Low	P1										
	Literature Review	Low	P2										
Full-Stack Development	Test Tool	Mid	FSD1										
Host-level Development	User Space	High	US1										
		Mid	US2										
	Kernel Space	Mid	KS1										
		Mid	KS2										
		High	KS3										
Device-level Development	Hardware Interface Layer (HIL)	High	HIL1										
	Flash Translation Layer (FTL)	Mid	FTL1										
Evaluation		Mid	E1										
Write a Paper		High	WP1										

* 중간 및 기말고사 기간은 일정 산출에서 제외
* FTL2 작업은 일단은 하지 않는 것으로 생각
* 2025년 04월 HotStorage에 내는 것을 목표로 함

참고 문헌

1. https://nvmexpress.org/specification/key-value-command-set-specification/
2. https://www.grandviewresearch.com/industry-analysis/cloud-computing-industry#:~:text=The%20global%20cloud%20computing%20market,transformative%20power%20of%20cloud%20computing.
3. https://www.techtarget.com/searchcloudcomputing/definition/noisy-neighbor-cloud-computing-performance
4. https://wiki.ubuntu.com/Kernel/Reference/IOSchedulers

E.O.D