

KV5 기술문서

목차

1. 기본 설계

- 1-1. 목표 특성과 환경
- 1-2. 전체 구조 개괄

2. 시스템 구조와 명령어

- 2-1. 물리적 구조와 제한
- 2-2. 논리적 구조와 제한
- 2-3. PEVFS 함수 특성
- 2-4. Shell 명령어 특성

3. 사용법 권고

- 3-1. 저장매체 특징
- 3-2. 사용 방법과 주의사항

4. 테스트 결과

- 4-1. 입출력 테스트
- 4-2. 기능 테스트
- 4-3. 속도/메모리 테스트

10진 접두어	K 킬로	M 메가	G 기가	T 테라	P 페타
크기	10^3 (.98 Ki)	10^6 (.95 Mi)	10^9 (.93 Gi)	10^{12} (.91 Ti)	10^{15} (.89 Pi)
2진 접두어	Ki 키비	Mi 메비	Gi 기비	Ti 테비	Pi 페비
크기	2^{10} (1.02 K)	2^{20} (1.05 M)	2^{30} (1.07 G)	2^{40} (1.10 T)	2^{50} (1.13 P)

1-1. 목표 특성과 환경

KVault5는 기존 KVault4의 취약점을 개선하고 편의성을 향상시킨 개인용 암호화 스토리지 프로그램이다. 외장하드 혹은 클라우드 저장소를 마운트하여 사용할 때 데이터 노출을 방지할 수 있다. KV5는 기존 파일시스템 위에서 저장폴더를 생성하고, 이는 비밀번호를 알아야 전용 프로그램으로만 읽고 쓸 수 있다. 따라서 외부에서 데이터를 확인하지 못하지만, NTFS/EXT4 등이 제공하는 파일 복구 기능을 사용할 수 있다.

기존 KV4의 문제점과 KV5의 개선 사항은 다음과 같다.

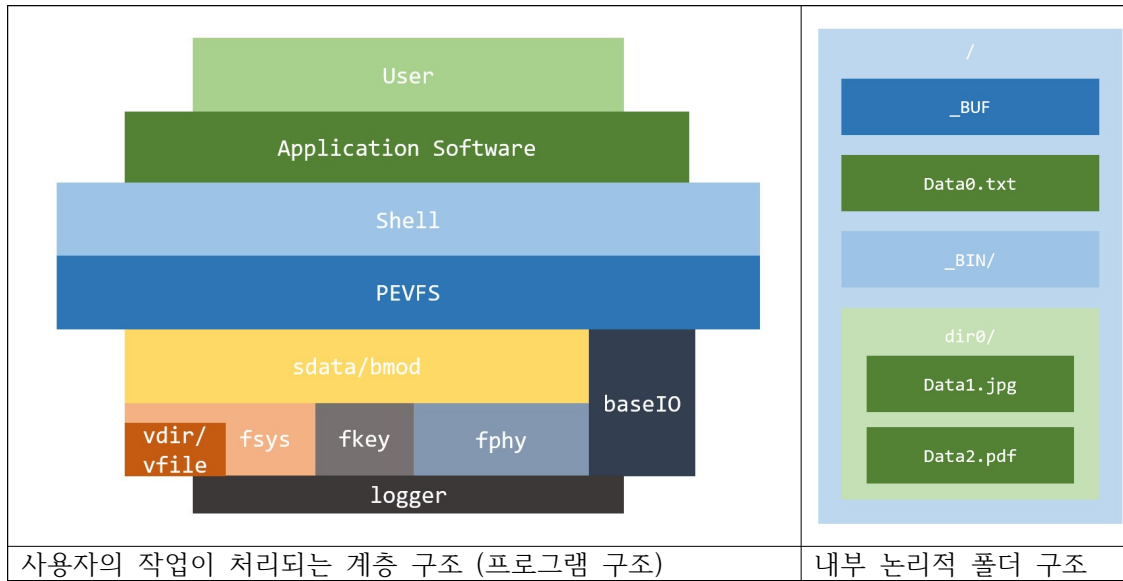
- 저속 원격 저장소에 지나치게 많이 접근하여 비효율적이었다. -> 데이터 입출력을 한 번에 묶어서 할 수 있게 되어 효율적이고 로컬 캐시를 사용하여 빠른 접근을 가능하게 하였다.
- 파일의 크기와 기록 시간 등 메타데이터가 노출된다. -> 블록화를 사용하여 전체 클러스터의 크기만을 추론할 수 있게 하였고, 변경 시간 등 메타데이터는 겹치게 하여 보호한다.
- 겹치는 이름이나 위계 문제 등 클러스터 보호가 미비하다. -> Private/Public 변수 분리와 동작 시 안전성 체크로 클러스터 운영을 더 안정적으로 할 수 있다.
- 입출력 중 저장소와 연결이 끊기면 모든 작업이 종료되고 재접속해야만 한다. -> 입출력 오류 처리 기능을 발전시켜 자동으로 재시도하고 세션이 유지된다.
- 동기적 동작으로 오랜 시간이 걸리는 작업의 진행도 확인 불가. -> 멀티스레딩을 활용하여 작업 진행 중에도 로그를 확인하거나 클러스터를 돌아볼 수 있다.
- 단일 사용자만을 고려한 설계. -> root 계정을 중심으로 하되, 일부 폴더만을 읽기전용으로 공유하거나 잠금을 거는 기능이 추가되어 더 자세한 사용자 제어가 가능해졌다.
- 비효율적인 자료구조와 제3자 프로그램 지원의 부재. -> 더 효율적인 자료구조를 사용했고 라이브러리로 제공함으로써 구조를 아는 다른 사람도 호환 프로그램 제작이 가능해졌다.
- 파이썬 단일스레드의 한계. -> 컴파일 언어인 Go를 사용하며 설계부터 동시성을 고려하여 속도와 메모리 효율성이 증대되었다.

다음과 같은 사용 패턴에 알맞도록 설계되었다.

- HDD/NAS/클라우드 등 원격 저장소는 입출력 속도가 매우 느리다. 반면 로컬 환경은 SSD와 최신 운영체제 등 빠른 데이터 처리가 가능하다.
- 클러스터는 대부분의 시간 동안 접속되지 않는 cold 상태로 보관되고, 가끔만 데이터의 입출력이 일어난다.
- 개인 수준의 폴더와 파일 수, 크기를 제어한다. 기업 환경과 같이 매우 큰 클러스터는 생각하지 않는다. (실용사용한도 : 폴더 10만, 파일 1000만, 전체크기 100TB 이내)
- 적당히 파일을 분배하여 저장한다. 주소할당이 4KB 미만의 작은 파일에는 비효율적이다. 또한 한 폴더 내부에 10만개 이상의 파일이 있다면 처리 속도가 저하될 수 있다.

	<pre> / _BIN/ _BUF files... </pre>
외부적으로는 암호화된 파일과 폴더이다.	내부적으로는 독립된 폴더 환경을 가진다.

1-2. 전체 구조 개괄



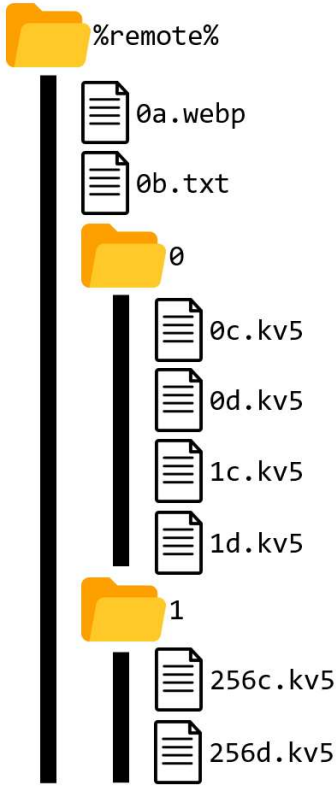
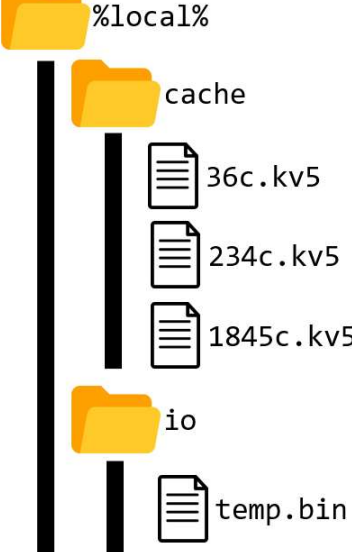

사용자(User)는 프로그램(Software)을 조작한다. 프로그램은 사용자의 조작에 따라 클러스터를 조작하는 명령을 생성하여 명령 해석기(Shell)에 입력한다. Shell은 현재 작업중인 폴더의 정보와 전체 클러스터 구조체(PEVFS)를 가지고 있는데, 입력받은 명령에 따라 PEVFS를 조작하거나 현재 정보를 업데이트한다. 이때 데이터 입출력 등 오랜 시간이 걸리는 작업은 백그라운드로 빠져 다음 명령을 빠르게 받을 수 있도록 한다.

PEVFS는 프로그램적으로는 하위에 sdata(setting data)와 bmod(basic module)를 가진다. sdata는 세션 키 등 설정 정보를 저장하고 bmod에는 클러스터 구동에 필요한 모듈이 위치한다. 작업이 진행되며 sdata/bmod의 데이터가 바뀌거나 해당 데이터를 기반으로 저장소와 입출력이 일어난다. baseIO는 오류없는 입출력을 담당한다. 만약 연결 끊김 등 문제가 발생하면, 자동으로 n초 대기 후 다시 읽기/쓰기 작업을 시도한다. logger는 작업 진행상황이나 오류 메시지를 기록하며, 최대 50개의 메시지를 저장할 수 있다.

PEVFS가 처리하는 작업은 논리적으로 fsys, fkey, fphy의 세 부분으로 구성되어 있다. fsys는 파일과 폴더의 논리적 위계와 인출 정보를 다룬다. 여기서 폴더는 실체가 없으며 저장소에 기록되지 않는다. 하위 폴더 혹은 파일에 대한 포인터를 가지는 메모리 상의 구조체로만 다뤄진다. 실체를 가지는 것은 오직 파일뿐인데, 파일은 암호화되어 그 데이터는 원격 저장소에 쪼개져 보관되고, 이름과 크기는 fsys에 보관된다. 또한 fptr이라는 고유번호를 부여받는데, fkey는 fptr과 파일의 암호화 키를 보관하고 관리한다. 일종의 해시맵과 같이 fptr-filekey를 효율적으로 저장하고 빨리 검색할 수 있게 한다. fphy는 물리적 저장소와의 입출력을 관리하는데, 하나의 큰 파일을 여러 개의 청크로 나눠 쓰고 다시 합쳐 읽는 과정을 제어한다. 청크로 분할되어 나눠 쓰인 파일은 암호화되었고 파편화되어서 공격자가 읽을 수 없게 된다.

내부적으로 KV5는 완전히 독립된 폴더 공간을 제공한다. 폴더는 항상 이름이 /로 끝나며, 파일과 폴더의 이름에 줄바꿈 문자(\n)는 들어갈 수 없다. 최상위 폴더는 /이며, 하위에 _BIN/이라는 폴더를 항상 가진다. 쓰기 파편화를 위해 _BUF라는 파일도 있는데, 이것은 PEVFS에 의해 자동으로 관리된다. 이외의 파일과 폴더는 유저가 자유롭게 읽거나 쓸 수 있으며 잠금 상태도 변경할 수 있다.

2-1. 물리적 구조와 제한

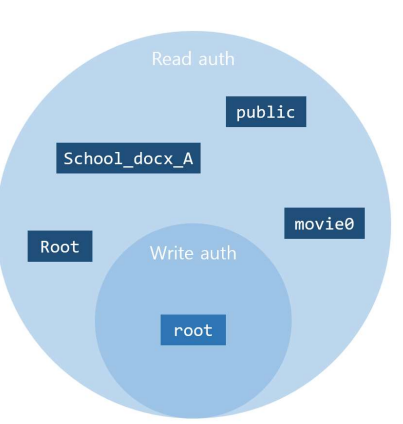
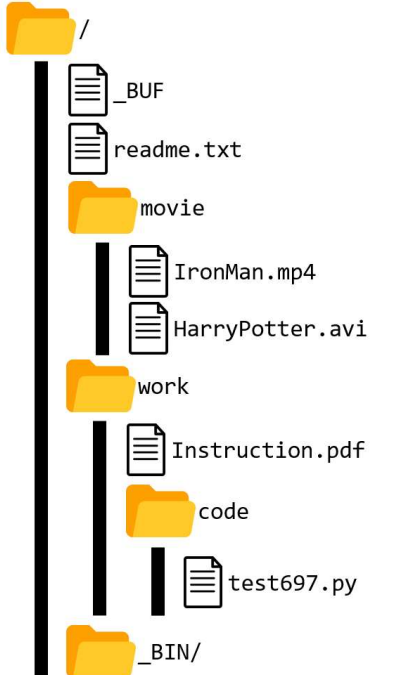
 <p>The diagram shows a directory structure for %remote%. It starts with a root folder %remote%. Inside, there are files 0a.webp and 0b.txt. Below them is a folder 0, which contains files 0c.kv5, 0d.kv5, 1c.kv5, and 1d.kv5. Below folder 0 is another folder 1, which contains files 256c.kv5 and 256d.kv5.</p>	<p>KV5의 저장소는 이미 인식된 드라이브 위에 파일과 폴더로써 구현된다. 원격 저장소는 ABCD 4종류의 파일로 구성된다.</p> <p>block A : account 파일. 로그인 정보와 클러스터 내부 파일에 대한 메타데이터가 암호화되어 저장된다.</p> <p>block B : basic 파일. 클러스터 전체의 크기와 이름, 쓰기 정보 등이 저장된다.</p> <p>block A/B는 한 클러스터에 하나씩 존재하며, 항상 정해진 위치와 이름으로 고정되어 있다.</p> <p>block C : control 파일. KV5의 파일은 청크 단위로 쪼개져 보관되는데, 각 청크들 사이 관계와 할당 정보가 암호화되어 저장된다. 크기는 384 KiB ($= 6B \times 65536$)로 고정되어 있다.</p> <p>block D : data 파일. 암호화되고 쪼개진 청크 데이터가 기록되는 곳이다. $Size_{chunk} \times 65536$의 크기로 고정되어 있다.</p> <p>block C/D는 unit 폴더 내부에 생성되는데, 한 unit 아래에 각각 256개씩 존재할 수 있다. unit은 총 65536개까지 존재할 수 있다.</p>
 <p>The diagram shows a directory structure for %local%. It starts with a root folder %local%. Inside, there is a folder cache, which contains files 36c.kv5, 234c.kv5, and 1845c.kv5. Below the cache folder is another folder io, which contains a file temp.bin.</p>	<p>로컬 운영체제가 설치된 빠른 드라이브에는 캐시파일과 입출력 대기 파일이 생성된다. 보통 로컬 폴더는 프로그램이 자동으로 생성과 삭제를 관리한다.</p> <p>cache 폴더에는 최대 512개의 block C가 존재할 수 있다. 만약 해당 블록을 읽을 상황이 발생한다면 느린 원격 저장소 대신 빠른 로컬 저장소에서 block C 파일을 읽을 수 있다. 로컬에 생성된 block C도 원격 저장소와 같이 암호화된 상태이다.</p> <p>io 폴더에는 block D로부터 나눠 쓰거나 읽을 파일이 위치한다. 나눠어진 청크들을 로컬 저장소에 하나의 파일로 모아야 암호화/복호화가 가능하다.</p>
 <p>The diagram shows a directory structure for %desktop%. It starts with a root folder %desktop%. Inside, there is a file Account.webp.</p>	<p>원격 저장소에 고정된 block A는 root 계정의 데이터를 담고 있다. root가 아닌 다른 읽기전용 계정의 데이터를 담은 block A는 임의의 위치에 존재할 수 있다.</p>

<div>Block A</div> <div>KSC header</div> <div>Account data (KDB text)</div> <div>Encrypted fsys data</div> <div>Encrypted fkey data</div> <div>Encrypted fphy data</div>	<p>block A는 KSC 형식을 따른다. 데이터 영역은 4개로 되어 있다. 0번 영역은 계정 이름과 로그인에 필요한 salt, pwhash, 암호화된 세션 키 등이 KDB 텍스트 형식으로 저장되어 있다.</p> <p>1번 영역은 암호화된 fsys 데이터로, 폴더와 파일 이름 / 크기 / 시간 정보가 기록되어 있다. 2번 영역은 암호화된 fkey 데이터이다. 원본 fkey 데이터는 5B fptr + 48B filekey가 반복되어 길이가 53의 배수이지만, 암호화된 결과는 그렇지 않다. 3번 영역은 암호화된 fphy 데이터이다. 처음 나오는 미할당 청크의 위치와 block C 암호 키가 저장되어 있다.</p>
<div>Block B</div> <div>Cluster data (KDB text)</div>	<p>block B에는 클러스터 관련 정보가 들어간다. 클러스터 이름, 청크 크기, 현재 블록 개수, 쓰기 id가 KDB 텍스트 형태로 저장된다.</p>
<div>Block C</div> <div>*65536, Encrypted</div> <div>Status byte</div> <div>Next address</div>	<p>클러스터에 저장된 파일은 청크로 나뉘는데, 각 청크는 block C에 고유 상태 바이트와 다음 청크의 주소를 가진다. 상태 바이트가 0이면 미기록 청크, 1~127이면 기록된 청크, 128이면 지울 예정인 청크, 129~255면 Garbage 청크이다.</p> <p>다음 청크의 주소는 5바이트인데, unit 주소 2바이트 + unit 내부 주소 1바이트 + 블록 내부 주소 2바이트로 구성된다. 따라서 청크 주소 (fptr 주소)는 $0 \sim 2^{40} - 1$의 값을 가질 수 있고, 블록 주소는 $0 \sim 2^{24} - 1$의 값을 가질 수 있다.</p>
<div>Block D</div> <div>*65536</div> <div>Chunk (fixed size)</div>	<p>block D는 한 unit당 256개씩 존재하고 블록 주소를 이름으로 가진다. unit 내부에 block C/D가 존재하기 때문에 한 unit당 파일의 최대 개수는 512개이다. KV5는 한 번에 최대 256개의 청크를 읽고 쓸 수 있다.</p> <p>청크 크기에 따라 블록 크기가 바뀐다. 청크 크기가 크다면 저장소에 더 많은 파일을 저장할 수 있고, 파일당 청크 분할 수가 작아져 입출력 속도가 향상된다. 하지만 하나의 청크는 하나의 파일에 대한 정보만을 가질 수 있어 저장되는 파일의 크기가 작다면 버려지는 공간 또한 많아진다.</p>

설정값	small	standard	large	default
청크 크기	4096 B (4Ki)	32768 B (32Ki)	262144 B (256Ki)	512 B
block D 크기	256 MiB	2048 MiB (2Gi)	16384 MiB (16Gi)	32 MiB
클러스터 한계	4 PiB	32 PiB	256 PiB	512 TiB

2-2. 논리적 구조와 제한

<p>vfile</p> <ul style="list-style-type: none"> Name (string) Time (int) Size (int) fptr (int) 	<p>모든 폴더와 파일 구조체는 메모리에 올라간다. 지나치게 많은 파일 수는 메모리에 부담이 될 수 있다.</p> <p>구조체 하나당 vfile은 40B, vdir은 80B를 차지한다. fkey와 암호화 등에 필요한 메모리를 고려하면 실질 메모리 사용량은 더 많다.</p>	<p>vdir</p> <ul style="list-style-type: none"> Name (string) Time (int) Locked (bool) subdir (vdir*) subfile (vfile*) 	<p>vdir의 name은 항상 /로 끝나며, vfile의 name은 /를 포함해서는 안 된다.</p> <p>잠금(lock)은 폴더의 접근을 제어하는데, PEVFS 옵션을 조정하면 잠기지 않은 폴더만 기록하거나 정보를 얻어오도록 할 수 있다.</p> <p>하위 폴더와 파일을 포인터(슬라이스)로 가져 전체 구조는 단방향 트리가 된다.</p>
---	--	---	--

	<p>계정(account)을 다양하게 설정하여 클러스터 접근과 데이터 조작 범위를 관리할 수 있다. 계정 이름은 대소문자를 구분하며, 특별한 계정(root)만 쓰기 작업이 가능하고 나머지는 읽기만 할 수 있다.</p> <p>접근 제어는 폴더 단위로 이루어지며, 현재 접근 가능한 폴더를 새로운 최상위 폴더(rootdir)로 하는 계정을 생성할 수 있다. 새 계정 생성 중 잠긴폴더 포함 여부를 False로 하면 잠기지 않은 폴더만 열람할 수 있는 계정이 만들어진다.</p>
	<p>root의 최상위 폴더는 항상 /이며, _BIN/ 폴더를 항상 하위에 가진다. root의 rootdir은 /를 이름으로 가질 수 있는 유일한 폴더이다. 또한 /과 /_BIN/은 항상 잠기지 않은 상태로 고정되어 있다.</p> <p>클러스터 내용의 일부만 공유하고 싶을 때, 새 계정을 만들 수 있다. 예를 들어 /movie/만을 포함하는 계정 “movie0”를 만들거나, /work/만을 포함하는 계정 “public”을 만들고 외부에 공유할 수 있다. 이 계정을 통해서만 제한된 범위의 읽기만이 가능하다.</p> <p>잠금을 사용하면 더 정밀한 제어가 가능하다. 예를 들어 /work/는 열림, /work/code/는 잠김이면, work 폴더를 열람할 수 있지만 code 폴더는 보지 못하는 계정을 생성할 수 있다.</p>

2-3. PEVFS 함수 특성

PEVFS는 클러스터를 조작하고 제어하는 기본 구조체이다. 모든 함수는 동기적으로 동작하기 때문에, 각 함수가 끝날 때까지 다른 함수를 실행시킬 수 없다. 구조체 외부의 클러스터 제어 함수 4개와 구조체 내부 데이터 제어 함수 24개가 존재한다.

클러스터 제어 함수는 전체 저장소와 관련된 기능을 수행한다.

PEVFS_New(remote string, cluster string, chunksize int) error
- 빈 원격 저장소 폴더 경로를 받아 새 클러스터를 생성한다.
PEVFS_Boot(desktop string, local string, remote string, blockApath string) (*PEVFS, [])byte, error)
- 클러스터 정보를 읽어 메모리에 올린다. 데이터는 암호화된 상태이다.
PEVFS_Exit(obj *PEVFS)
- 내부 민감정보를 지우고 PEVFS를 정지 상태로 설정한다. 로컬 폴더를 지운다.
PEVFS_Rebuild(remote string, pw []byte, kf []byte) error
- block C와 fphy 데이터를 재작성한다. 원격 저장소의 모든 block C를 덮어쓴다.

데이터 제어 함수 A는 PEVFS의 전반적인 작업 수행을 제어한다.

Abort(reset bool, abort bool, working bool) (bool, bool)
- PEVFS의 정지/작업 플래그를 재설정하거나 가져온다.
Debug() ([]int, [][]byte)
- PEVFS 디버그 정보와 내부 민감정보의 일부를 반환한다.
Log(reset bool) string
- 로그 데이터를 삭제하거나 읽어온다.

데이터 제어 함수 B는 계정 로그인과 생성 등 계정 관련 기능을 제공한다.

Login(pw []byte, kf []byte, sleeptime int) error
- 로그인하고 메모리의 데이터를 복호화한다. root 계정이라면 헤더를 재작성한다.
AccReset(pw []byte, kf []byte, hint []byte) error
- 현재 계정의 비밀번호, 키 파일, 비밀번호 힌트를 바꾸고 새 헤더로 재작성한다.
AccExtend(pw []byte, kf []byte, hint []byte, account string, wrlocked bool) (string, error)
- 현재 작업 폴더를 rootdir로 하는 읽기전용 계정을 바탕화면에 파일로 생성한다.

데이터 제어 함수 C는 클러스터 탐색과 현재 작업 폴더 이동 등 가벼운 기능을 가진다.

Search(name string) []string
- 현재 작업 폴더 하위에 특정 패턴을 가진 파일/폴더 이름이 있는지 검색한다.
Print(wrlocked bool) string
- 현재 작업 폴더와 하위 내용을 텍스트로 생성한다. (디버깅용)
Teleport(path string) bool
- /로 끝나는 전체 폴더 경로를 받아 현재 작업 폴더의 위치를 바꾼다.
NavName() ([]string, []bool)
- 현재 작업 폴더의 직접 하위 항목의 이름과 잠김 여부만을 가져온다.
NavInfo(wrlocked bool) ([]string, []int, []int, []int, []int)
- 현재 작업 폴더의 직접 하위 항목의 이름, 크기, 시간 등 모든 정보를 가져온다.

데이터 제어 함수 D는 외부 환경과의 데이터 입출력을 담당한다.

ImBin(name string, data []byte) error
- 현재 작업 폴더 아래에 data 내용의 name 파일을 만든다. 만약 name 파일이 이미 존재 한다면 data 내용으로 교체된다.
ImFiles(paths []string) error
- 외부 파일 경로를 받아 현재 작업 폴더 아래에 파일을 가져온다. 동일한 이름의 파일은 교체된다.
ImDir(path string) error
- 외부 폴더 경로를 받아 현재 작업 폴더 아래에 폴더와 그 하위 항목을 가져온다. 동일한 이름의 폴더가 존재한다면 오류가 발생한다.
ExBin(name string) ([]byte, error)
- 현재 작업 폴더 아래에 name 파일의 내용을 반환한다. ImBin/ExBin은 데이터가 내부 적으로 처리되어 평문 파일 내용이 로컬 환경에 기록되지 않는다.
ExFiles(names []string) error
- 현재 작업 폴더 아래의 name 파일을 로컬 환경으로 읽어온다. %desktop%/kv5export/ 폴더가 생성되고 파일은 이 폴더 내부에 생성된다.
ExDir(path string) error
- 전체 폴더 경로를 입력받아 폴더와 그 하위 항목을 로컬 환경으로 읽어온다. %desktop%/kv5export/ 폴더가 생성된다.

데이터 제어 함수 E는 클러스터 내부 폴더와 파일의 조작을 담당한다.


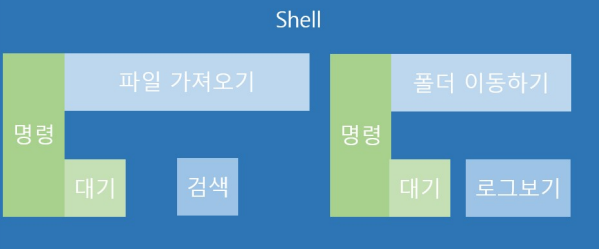
Delete(paths []string) error
- 전체 파일/폴더 경로를 입력받아 해당 항목을 fsys/fkey 상에서 지우고 fphy에서는 할당 해제한다. /과 /_BIN/은 삭제할 수 없다.
Move(names []string, dst string) error
- 현재 작업 폴더 아래의 name 폴더/파일을 목적지 폴더로 이동시킨다. dst는 전체 경로 여야 하며, 상위 폴더를 하위 폴더 밑으로 이동시키는 위계 문제가 발생하면 오류가 반환된다.
Rename(before []string, after []string) error
- 현재 작업 폴더의 before 폴더/파일의 이름을 after로 바꾼다. /_BIN/은 이름을 바꿀 수 없고, 다른 폴더와 파일도 겹치는 이름으로 설정할 수 없다.
DirNew(names []string) error
- 현재 작업 폴더 아래에 name 폴더를 생성한다. 겹치는 이름으로는 생성할 수 없다.
DirLock(path string, islocked bool, sub bool) error
- 폴더 전체 경로를 받아 잠금 상태를 변경한다. sub가 True면 하위 폴더도 변경된다.

데이터 제어 함수 F는 클러스터의 무결성 확인과 오류 정정 기능을 제공한다.

CluCheck() (int, error)
- 현재 작업 폴더 하위의 파일들에 대해, fkey/fphy(block C) 상에서 제대로 기록되었는지 검증하고 문제가 있는 파일의 개수를 반환한다.
CluRestore(rename bool, rewrite bool, rebuild bool) (int, error)
- 클러스터 오류를 정정한다. “잘못된 이름 정정”, “미삭제 체크 재기록”, “전체 파일 무결성 검증” 작업을 할 수 있다. 정정되거나 재기록된 개수를 반환한다.

2-4. Shell 명령어 특성

Shell은 PEVFS 상위에 위치하며, 조작을 간편하고 효율적으로 할 수 있도록 도와준다. 입출력 작업 등 시간이 오래 걸리는 작업은 백그라운드로 빠져 빠르게 다음 명령에 응답할 수 있다. Shell은 오직 하나의 public 함수만 제공하며, Shell public 필드와 함수를 이용해 클러스터를 제어할 수 있다. (PEVFS 외부 4개, PEVFS 내부 24개, Shell 1개, Test 3개 - KV5에서는 총 32개의 public 함수가 제공된다.)

	<p>PEVFS는 한 작업이 끝나야 다음 작업을 할 수 있다. 파일 입출력 등 오래 걸리는 작업이 진행되는 동안 아무 것도 할 수 없다.</p>
	<p>Shell은 백그라운드 작업을 사용하므로 입출력 도중에 검색이나 로그 보기와 같이 간단한 작업을 동시에 처리할 수 있다.</p>

Shell의 Command 함수는 명령과 옵션을 입력받아 작업을 수행하고 처리 결과를 반환한다. 명령은 NOP 제외 28개가 가능하다. 옵션 특성은 다음과 같다. T : “true”, F : “false”, N : index of Shell.CurName, ~ : series with len 1+, > : return value. Async 요소가 BG(background)면 백그라운드 실행이 가능하고, FG(foreground)면 백그라운드 명령 실행 중 동시에 작동할 수 있다. explain order option IOstr IObyte Async 순이다.

Shell 초기화	init	flagsz[TF]	.	.	.
새 클러스터 생성	new	remote cluster csize[small standard large *]	.	.	.
클러스터 읽기	boot	desktop local remote blockApath	cluster[> account[>]	hint[>]	.
모든 작업 종료	exit	.	.	.	FG
fphy 재설정	rebuild	remote	.	password keyfile	.

제어 플래그 관리	abort	reset[TF] abort[TF] working[TF]	.	.	FG
디버그 정보 보기	debug	countLck[TF]	info0[> info1[> info2[> info3[>	.	FG
로그 정보 보기	log	reset[TF]	logdt[>	.	FG
클러스터 로그인	login	sleep[1 10 30 60 *]	.	password keyfile	.
비밀번호 재설정	reset	.	.	password keyfile hint	.
새 계정 생성	extend	account wrlock[TF]	newpath[>	password keyfile hint	.

이름패턴 검색	search	name	result[>	.	FG
현재 폴더 출력	print	wrlock[TF]	result[>	.	FG
하위폴더 이름 보기	navigate	path	subdir[>	.	FG
상태 새로고침	update	.	.	.	FG

이진파일 가져오기	imbin	name	.	data	BG
파일들 가져오기	imfile	fullpath[~]	.	.	BG
폴더 가져오기	imdir	fullpath	.	.	BG
이진파일 내보내기	exbin	name[~]	.	data[>][~]	BG
파일들 내보내기	exfile	name[~]	.	.	BG
폴더 내보내기	exdir	fullpath	.	.	BG

하위항목 삭제	delete	nameN[N][~]	.	.	BG
하위항목 이동	move	tgtdir nameN[N][~]	.	.	BG
하위항목 재명명	rename	nameN[N][~]	newname[~]	.	BG
하위폴더 생성	dirnew	name[~]	.	.	BG
잠금 변경	dirlock	lock[TF] nameN[N][~]	.	.	BG

현재 무결성 체크	check	.	result[>	.	BG
전체 무결성 복구	restore	mode[rename rewrite rebuild]	result[>	.	BG
IObuf 초기화	*	.	.	.	BG

3-1. 저장매체 특징

RAM (주기억장치) : RAM은 가장 속도가 빠르며, 컴퓨터가 구동에 필요한 모든 정보를 올려두는 곳이다. 전원을 끄면 모든 내용이 사라지지만, 공격자가 전원이 켜진 상태의 컴퓨터를 입수한다면 메모리의 내용이 유출될 수 있다. KV5의 경우, RAM 내용이 유출되어 프로그램의 메모리 상태가 알려진다면 비밀번호, 파일명, 파일 내용 등 모든 데이터가 노출될 수 있어 주의가 필요하다.

HDD (하드디스크/외장하드) : 비교적 느린 속도로 동작하는 보조기억장치이다. HDD는 자기장을 사용하여 물리적으로 정보를 저장하기에, 공격자가 지워진 정보를 쉽게 복구할 수 있다. HDD에 한 번이라도 쓰여진 정보는 다시 복구될 수 있다고 간주하고 사용해야 한다. HDD의 정보를 안전하게 지우려면 단순히 포맷하는 것이 아니라, 모든 용량을 다른 데이터로 채워야 지워야 한다. (1TB 하드면 1TB만큼 다른 정보를 채워야 지워짐)

USB : USB는 플래시메모리를 사용해 전자적으로 정보를 저장하는 반도체이다. 모든 보조기억장치에 대해, 운영체제는 파일을 지울 때 실제로 데이터를 지우는 것이 아니라, “지움 표시”만 해놓는다. 따라서 지워진 파일을 다시 복구시킬 수 있다. USB 또한 HDD와 같이 사용하는 것이 바람직하다.

SSD (반도체 드라이브/외장SSD) : USB와 마찬가지로 플래시메모리로 되어 있다. USB는 HDD보다도 느리지만, SSD는 빠르게 정보를 읽고 쓸 수 있는데, 이것은 플래시메모리를 병렬화해서 입출력하기 때문이다. SSD는 여러 메모리 셀에 동시에 데이터를 나눠 저장하기에, 내부 구조가 복잡하고 공격자가 파일을 복구하기가 어렵다. 특히 TRIM으로 내부상태를 재배치하면 데이터 복구는 사실상 불가능하다. 운영체제에 따라 다르지만, 보통 TRIM 주기는 일주일 정도이다.

SSD (OS installed) : 운영체제가 설치된 드라이브는 다른 드라이브와 다르게 매우 취약하다. OS는 동작하면서 수많은 캐시파일을 남기고, 사용자가 어떤 행동을 했고 어떤 파일을 열었는지 이러한 흔적들로 쉽게 추적 가능하다. OS가 설치된 저장장치는 더 특별한 보호가 필요하다.

클라우드 : Google, Naver 등에서는 클라우드 스토리지 서비스를 제공한다. 적절한 소프트웨어를 사용하면 클라우드를 USB 인식하듯이 마운트할 수 있다. 클라우드에 저장된 내용은 비밀번호가 없는 타인은 확인이 불가하나, 서비스 제공 업체는 얼마든지 접근할 수 있다. 또한 OS 데이터가 유출된다면, 그곳에 남아있는 정보를 조합해 클라우드 비밀번호를 추론하는 것도 가능하니 주의해야 한다.

NAS : NAS는 개인이 직접 저장장치와 미니컴퓨터로 구축한 사설 클라우드이다. 클라우드 서비스 업체가 데이터를 들여다볼 위험은 없지만, 서버 설정이 올바르지 못한 경우 해킹이나 데이터 유출의 위험이 있다. 또한 서버에 직접 접근하는 물리적 공격에 대해서는 일반 외장하드와 같이 취약하다는 점을 주의해야 한다.

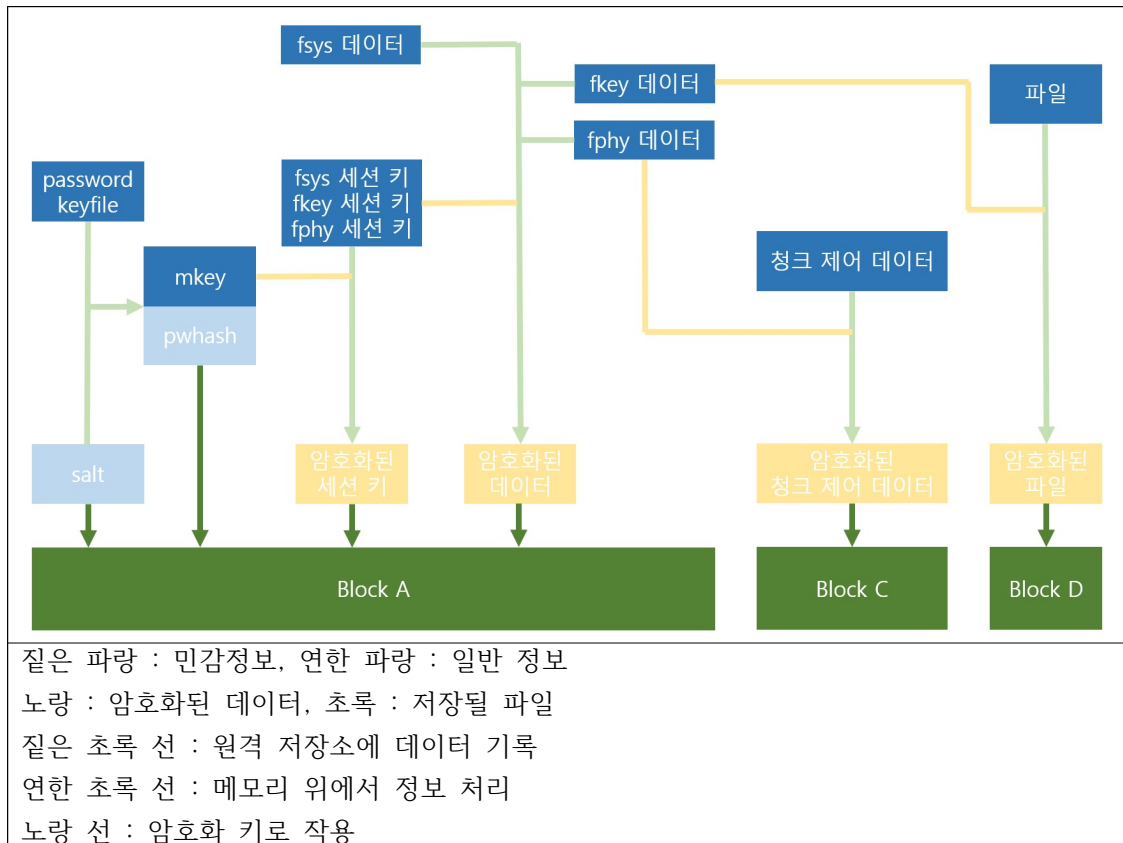
3-2. 사용 방법과 주의사항

KV5 사용 시 데이터 유출을 방지하기 위해 정해진 절차를 따르는 것을 권장한다. OS가 설치된 드라이브는 다른 보호 수단으로 보호되고 있으며, 원격 저장소는 HDD를 사용하여 한번 기록된 데이터는 삭제하기 힘든 상황으로 가정한다. (HDD에는 항상 암호화된 정보만 써야 한다. 평문 데이터를 그대로 기록해서는 안 된다.)

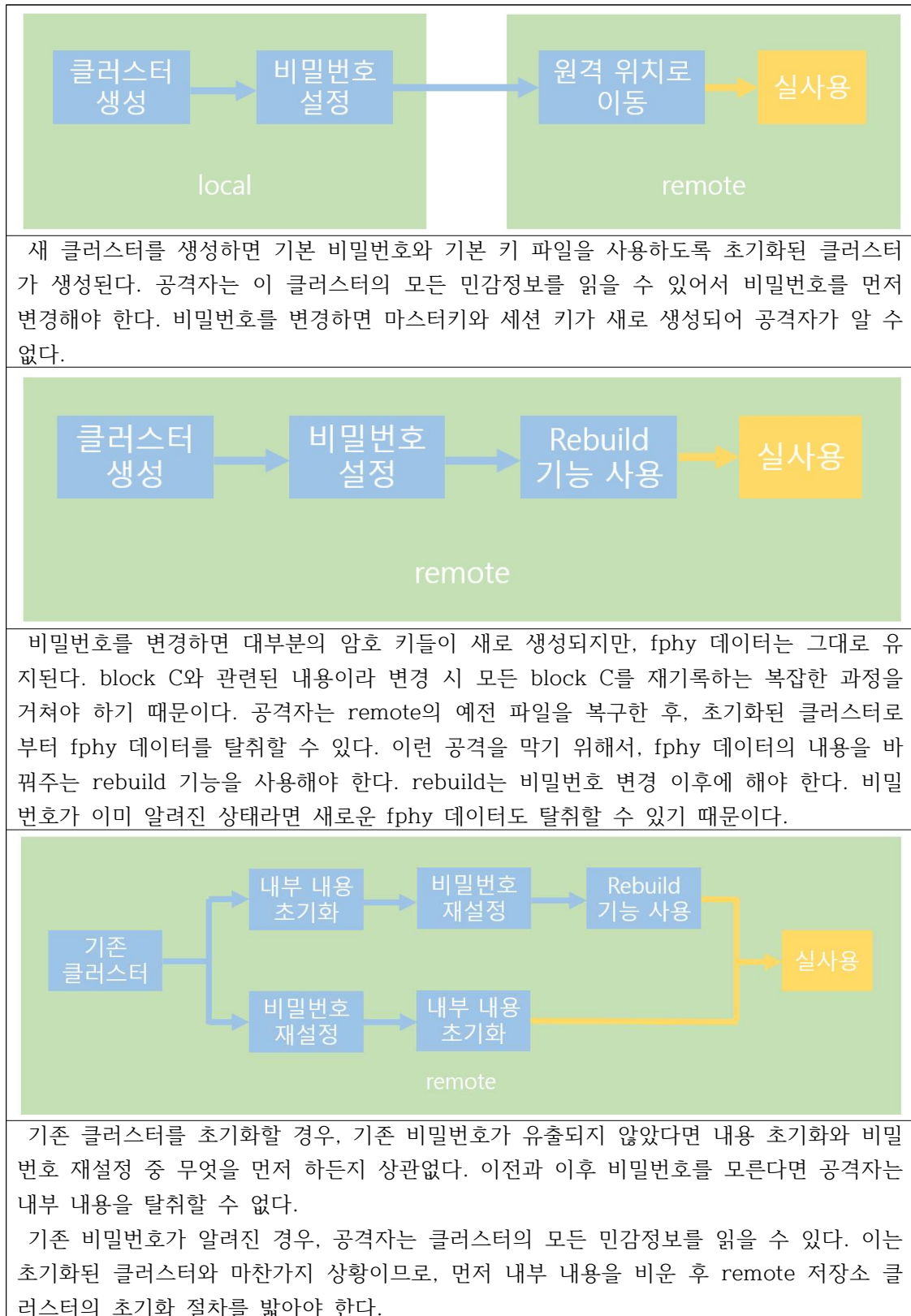
KV5에서 다루는 민감정보는 다음과 같은 종류가 있다.

- 필드 이름	- 역할	- 저장 위치	- 위험 상황
pw, keyfile	mkey, pwhash 유도	메모리 (외부에서 공급)	신규 생성시 초기화
salt	키 유도 과정에 사용	block A (공개)	.
mkey	세션 키 암호화	메모리	pw, keyfile 유출
pwhash	pw, keyfile 검증	block A (공개)	해시 역함수 계산
fsys 세션 키	fsys 데이터 암호화	block A (mkey 잠금)	mkey[0:48] 유출
fkey 세션 키	fkey 데이터 암호화	block A (mkey 잠금)	mkey[48:96] 유출
fphy 세션 키	fphy 데이터 암호화	block A (mkey 잠금)	mkey[96:144] 유출
fsys 데이터	논리 위계구조 저장	block A (세션 키 잠금)	fsys 세션 키 유출
fkey 데이터	파일 키 저장	block A (세션 키 잠금)	fkey 세션 키 유출
fphy 데이터	block C 암호화	block A (세션 키 잠금)	fphy 세션 키 유출
block C 데이터	청크 관계 저장	block C (fphy 잠금)	fphy 데이터 유출
block D 데이터	파일 데이터 저장	block D (filekey 잠금)	fkey 데이터 유출

KV5 구동에 필요한 정보들은 다음 관계로 처리되어 저장된다.



다음은 안전하게 클러스터를 초기화하는 방법 3가지이다.



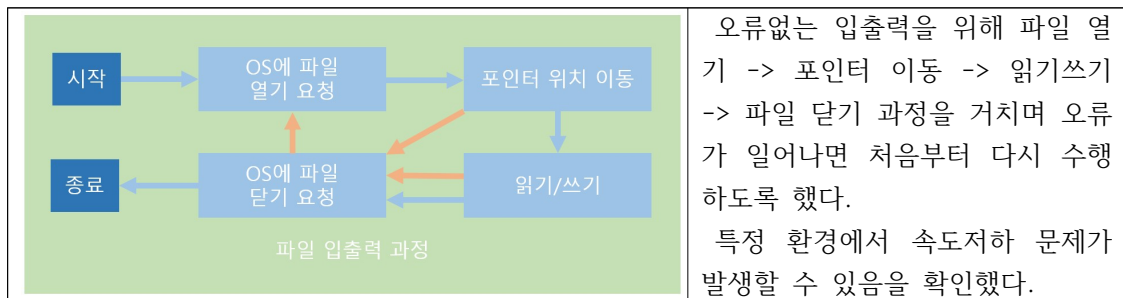
4-1. 입출력 테스트

클러스터 생성, 로그인, 파일 입출력의 기본 기능을 시험했다.

- 테스트 종류	- 대상 파일 크기	- 원격 드라이브 종류	- 결과
클러스터 생성	256 MiB	HDD	성공
파일 입력	10 GiB	HDD	성공
파일 출력	10 GiB	HDD	성공
클러스터 생성	32 MiB	클라우드	실패 (사용자 간섭 추정)
로그인	12 KiB	클라우드	성공
파일 입력 1차	1 MiB	클라우드	실패 (전송 오류 추정)
파일 입력 2차	1 MiB	클라우드	성공
파일 출력	1 MiB	클라우드	성공

KV5의 오류 정정 입출력 기능을 시험하기 위해 데이터 입출력 도중 외장하드 연결을 해제한 후 다시 연결하는 테스트를 진행했다.

- 테스트 종류	- 대상 파일 크기	- 원격 드라이브 종류	- 결과
클러스터 생성	256 MiB	HDD	성공
파일 입력	10 GiB	HDD	성공
파일 출력 1차	10 GiB	HDD	부분실패 (일부데이터 오류)
파일 출력 2차	10 GiB	HDD	성공



4-2. 기능 테스트

KV5의 public 함수가 제대로 동작하는지 시험하였다.

- 테스트 종류	- 원격 드라이브 종류	- 결과
PEVFS 외부 함수 4개	HDD	성공
PEVFS 내부 함수 24개	HDD	성공
Shell Command	HDD	성공
Test 함수 3개	HDD	성공

추가로 KV5가 제대로 구현되었는지 검증하기 위해 python 읽기 코드를 만들어서 시험하였다. python 코드와 코드가 사용하는 라이브러리는 go 런타임에 의존적이지 않고, 논리적 호환성을 갖춘 독립적인 코드가 되게끔 작성했다. 시험 결과 KV5 클러스터가 의도에 맞게 작성되었음을 확인했다.

4-3. 속도/메모리 테스트

Basic 테스트는 클러스터 생성 속도와 4GiB 파일 읽기/쓰기 속도를 측정한다.

원격 저장소 종류	청크 크기 설정	의존성	클러스터 생성 시간 (s)	클러스터 생성 메모리 (MiB)
SSD	standard	.	2.77	1000
HDD	standard	.	35.4	1000
원격 저장소 종류	파일 읽기 속도 (MiB/s)	파일 읽기 메모리 (MiB)	파일 쓰기 속도 (MiB/s)	파일 쓰기 메모리 (MiB)
SSD	4654.5	100	1259.9	100
HDD	71.3	100	68.2	100

IO 테스트는 클러스터 로그인, 4GiB 파일 내보내기 / 가져오기 속도를 측정한다.

원격 저장소 종류	청크 크기 설정	의존성	클러스터 로그인 시간 (s)	클러스터 로그인 메모리 (MiB)
SSD	standard	Test_Basic	1.74	1100
HDD	standard	Test_Basic	2.16	1100
원격 저장소 종류	파일 내보내기 속도 (MiB/s)	파일 내보내기 메모리 (MiB)	파일 가져오기 속도 (MiB/s)	파일 가져오기 메모리 (MiB)
SSD	445.5	1000	346.7	1500
HDD	695.5	1000	27.5	1500

Multi 테스트는 실용사용한도 상황에서 block A 입출력 성능을 평가한다. KV5는 이 한도를 넘는 클러스터도 다룰 수 있지만, 속도나 메모리 문제로 권장하지는 않는다. standard 청크를 사용하는 클러스터의 실용사용한도는 “폴더 10만, 파일 1000만, 전체체크 100TB 이내”이다.

원격 저장소 종류	청크 크기 설정	의존성	상황 설정 시간 (s)	상황 설정 메모리 (MiB)
SSD	standard	.	4.37	1900
HDD	standard	.	4.47	1900
원격 저장소 종류	block A 읽기 시간 (s)	block A 읽기 메모리 (MiB)	block A 쓰기 시간 (s)	block A 쓰기 메모리 (MiB)
SSD	7.03	14700	4.60	11500
HDD	37.63	15300	19.35	12000

