Zen Crypted Operating System

Технічні характеристики

Порівняльний аналіз архітектур I/O у технологіях віртуалізації: VirtIO, Xen та VMware ESXi

На основі Технічних Вимог

Згідно з Постановою КМУ № 205 від 21.02.2025

Формалізованих за стандартом ISO/IEC/IEEE 29148:2018

Порівняльний аналіз архітектур I/O у технологіях віртуалізації: VirtIO, Xen та VMware ESXi

Аналітичний огляд технічних характеристик і механізмів

19 серпня 2026 р.

Зміст

Анотація

У статті проведено комплексний науково-технічний аналіз архітектур введення-виведення (I/O) трьох провідних рішень у сфері віртуалізації: фреймворку VirtIO (стандарт OASIS), гіпервізору Xen та комерційної платформи VMware ESXi. Особливу увагу приділено взаємодії між компонентами Frontend та Backend, механізмам організації спільної пам'яті (virtqueues / ring buffers / grant tables), керуванню перериваннями, zero-copy шляхам даних, оптимізаціям interrupt coalescing / NAPI / event mitigation, а також паспортним тактико-технічним характеристикам мережевих і дискових підсистем. Результати дослідження систематизовано у вигляді детальних порівняльних таблиць і якісного аналізу latency/IOPS-шляхів. Показано, що сучасна конкуренція змістилася з CPU-overhead віртуалізації на рівень ефективності периферійного I/O під високими навантаженнями (High IOPS, Low Latency, multi-queue scalability).

1. Вступ

Сучасні хмарні обчислення, центри обробки даних (ЦОД) та edge-системи вимагають мінімальних накладних витрат на віртуалізацію периферійних пристроїв. Історично повна емуляція апаратного забезпечення (full device emulation) створювала суттєве навантаження на центральний процесор через постійні перемикання контексту (VM-Exits / VM-Entries) при кожній операції з регістрами або чергами пристрою. Для вирішення цієї проблеми було розроблено концепцію паравіртуалізації (paravirtualization), де гостьова операційна система взаємодіє з гіпервізором через оптимізовані драйвери, які «знають», що працюють у віртуальному середовищі.

Ключовим архітектурним патерном став розподіл підсистеми I/O на два компоненти:

- **Frontend** — драйвер у гостьовій ОС (guest kernel);
- **Backend** — компонент на стороні гіпервізора / хоста (або привілейованого домену).

Метою цієї роботи є детальне порівняння внутрішньої архітектури рівнів Frontend/Backend, механізмів спільної пам'яті, сповіщень, zero-copy шляхів і паспортних характеристик дискових і мережевих підсистем для VirtIO (KVM/QEMU), Xen та VMware ESXi. Акцент зроблено на інженерних деталях, які визначають реальну продуктивність під високими навантаженнями.

2. Архітектура взаємодії Frontend та Backend

Розділення підсистеми введення-виведення на Frontend і Backend є фундаментальним принципом сучасного паравіртуалізованого I/O. Нижче розглянуто кожну технологію з акцентом на структури даних і життєвий цикл запиту.

2.1. VirtIO (OASIS Standard)

VirtIO визначає стандартизований інтерфейс між гостьовою ОС і гіпервізором. Основним механізмом bulk data transport є *virtqueue* (також званий vring). Кожен пристрій може мати до 65 536 virtqueues. Кожна virtqueue складається з трьох областей у спільній пам'яті гостя:

1. **Descriptor Area (Descriptor Table)** — масив дескрипторів. Кожен дескриптор містить:
   - `addr` — фізичну адресу (GPA) буфера в пам'яті гостя;
   - `len` — довжину буфера;
   - `flags` — `VRING_DESC_F_NEXT`, `VRING_DESC_F_WRITE`, `VRING_DESC_F_INDIRECT`;

- `next` — індекс наступного дескриптора (для ланцюжків scatter-gather).

2. **Driver Area (Available Ring)** — кільце, куди драйвер (гость) записує індекси готових дескрипторів. Містить поля `flags`, `idx` і масив `ring[]`.
3. **Device Area (Used Ring)** — кільце, куди пристрій (хост) записує індекси оброблених дескрипторів разом із кількістю записаних байтів.

Життєвий цикл типового запиту (split virtqueue):

1. Гость виділяє дескриптори, заповнює `addr/len/flags`, додає індекс голови ланцюжка в Available Ring, інкрементує `avail.idx` (з memory barrier).
2. Гость виконує kick (MMIO write або hypercall) — це може викликати VM-Exit.
3. Backend читає Available Ring, обробляє буфери (часто через DMA або zero-copy), записує результат у Used Ring, інкрементує `used.idx`.
4. Backend надсилає notification (vIRQ / MSI-X), яку гость обробляє (часто через NAPI або adaptive polling).

Сучасні реалізації підтримують *packed virtqueues* (VirtIO 1.1+), які зменшують кількість memory accesses і latency порівняно зі split-форматом. Також широко використовується `vhost` / `vhost-user` + DPDK для винесення datapath у userspace і мінімізації переходів у kernel.

2.2. Xen Split Driver Model

Xen використовує класичну модель split drivers з ізольованими доменами. Frontend (`xen-blkfront`, `xen-netfront`) працює в DomU, Backend (`xen-blkback`, `xen-netback`) — зазвичай у Dom0 (або окремому driver domain / stubdom).

Ключові механізми:

- **Shared Ring Buffers** — односторінкові (або кілька) кільцеві буфери запитів/відповідей, розміщені у спільній пам'яті.
- **Grant Tables** — механізм, за допомогою якого DomU надає Dom0 тимчасові права доступу до конкретних сторінок пам'яті. Grant reference передається через ring; Backend виконує `gnttab_map_refs` (або використовує persistent grants).
- **Event Channels** — асинхронний механізм сповіщень (віртуальні IRQ), що замінює класичні переривання.

Grant Tables забезпечують сильну ізоляцію: Backend ніколи не отримує прямий доступ до всієї пам'яті гостя, лише до явно наданих сторінок. Це підвищує безпеку, але додає накладні витрати на map/unmap (TLB flushes, hypercalls). Persistent grants (введені для зменшення overhead) дозволяють повторно використовувати одні й ті самі grant mappings, суттєво підвищуючи IOPS при багатьох concurrent guests.

2.3. VMware ESXi Mono-Framework

VMware використовує пропрієтарні паравіртуальні драйвери (VMXNET3 для мережі, PVSCSI для диска), які взаємодіють безпосередньо з монолітним VMkernel. Спільна пам'ять реалізована у вигляді Shared Ring Buffers (Request / Completion / Message rings для PVSCSI). Адресація виконується через апаратні таблиці другого рівня (Intel EPT / AMD NPT). Сповіщення — через VMM backdoor (`VMCALL`) або virtual MSI-X.

Архітектура більш «монолітна»: немає проміжного Dom0, datapath оптимізований усередині VMkernel. Це дає високу передбачуваність і стабільність у enterprise-середовищах, але меншу відкритість і гнучкість порівняно з VirtIO.

Детальне порівняння параметрів архітектури наведено в таблиці 1.

Табл. 1: Порівняльний аналіз архітектури інфраструктури I/O (Backend/Frontend)

Параметр	VirtIO (KVM/QEMU)	Xen Hypervisor	VMware ESXi
Архітектура I/O	VirtIO Framework (OASIS)	Split Driver Model	VMware PV Drivers
Frontend	`virtio-blk/net/scsi`	`xen-blkfront`, `xen-netfront`	`vmxnet3`, `pvscsi`
Backend	`vhost-*` / QEMU	`xen-blkback`, `xen-netback`	VMkernel (engine)
Спільна пам'ять	Virtqueues (split/packed)	Ring Buffers + Grant Tables	Shared Ring Buffers
Трансляція адрес	GPA + vIOMMU / direct	Grant Tables (map/unmap)	Intel EPT / AMD NPT
Сповіщення Fr→Bk	Kick (MMIO / hypercall)	Event Channel	VMM Backdoor / VMCALL
Сповіщення Bk→Fr	vIRQ (MSI-X)	Event Channel	Virtual MSI-X
Оптимізація IRQ	NAPI / Adaptive Polling	Event Mitigation	Interrupt Coalescing
Bypass / high-perf	vhost-user + DPDK	Stubdoms / SR-IOV	DirectPath I/O / SR-IOV / UPT
Ізоляція пам'яті	Залежить від гіпервізора	Сильна (grants)	EPT + IOMMU

3. Механізми спільної пам'яті та zero-copy

3.1. VirtIO: Descriptor chains і zero-copy

VirtIO дозволяє будувати ланцюжки дескрипторів (scatter-gather). Backend може читати/писати безпосередньо в гостьову пам'ять без проміжного копіювання (zero-copy), якщо використовується vhost або апаратний DMA з правильним IOMMU mapping. При використанні QEMU userspace backend можливе додаткове копіювання; vhost-net / vhost-blk усуває його.

Packed virtqueues додатково зменшують кількість memory barriers і cache-line bouncing.

3.2. Xen: Grant Tables і copy vs map

Класичний шлях Xen часто включає copy (особливо для мережі з small packets). Persistent grants і grant mapping дозволяють Backend виконувати DMA безпосередньо в гостьові сторінки. Однак map/unmap операції та TLB flushes залишаються джерелом overhead, особливо при високій concurrency і великій кількості VIF/VBD.

3.3. VMware: Shared rings + EPT

VMware покладається на EPT для ефективного доступу до гостьової пам'яті. PVSCSI і VMXNET3 використовують shared rings для дескрипторів команд, а дані передаються через virtual scatter-gather DMA. Overhead мінімізований за рахунок глибокої інтеграції у VMkernel.

4. Керування перериваннями та сповіщеннями

Усі три системи намагаються зменшити кількість переривань:

- **VirtIO** — прапорці `VIRTQ_AVAIL_F_NO_INTERRUPT` / `VIRTQ_USED_F_NO_NOTIFY`, NAPI, adaptive polling, interrupt coalescing на рівні MSI-X.
- **Xen** — Event Channel mitigation, batching сповіщень.
- **VMware** — Interrupt Coalescing у VMXNET3 / PVSCSI, можливість налаштування coalescing parameters.

При дуже високих packet rates / IOPS усі системи переходять у polling mode (або hybrid), щоб уникнути interrupt storm.

5. Тактико-технічні характеристики мережевих підсистем

Табл. 2: Паспортні ТТХ мережевих підсистем

Технічний параметр	VirtIO-Net (KVM)	Xen Netfront	VMware VMXNET3
Пропускна здатність	Обмежена RAM/CPU / DPDK	Обмежена Dom0 CPU	Обмежена hardware хоста
Максимальний MTU	До 9000 (jumbo)	До 9000	До 9000
Множинні черги	До 256 (multi-queue)	До 8 (типово)	До 8–32 (RSS)
Розмір черги	256–4096 (конфіг.)	Фіксований ∼256–512	Rx 256–4096, Tx 512–4096
Апаратне розвантаження	TSO/GSO/UFO/CSUM, RSS	TSO/GSO/CSUM	TSO/LRO/CSUM/RSS/VXLAN/Geneve
Низька затримка	DPDK / vhost-user	SR-IOV / VF	DirectPath I/O / SR-IOV / UPT
Zero-copy path	vhost + DPDK	Persistent grants + map	Shared rings + EPT

VirtIO + DPDK/vhost-user демонструє найвищу scalability за кількістю черг і найкращий path для userspace networking. VMware VMXNET3 має найбагатший набір offloads (включаючи tunnel offloads), що критично для NSX / overlay networks. Xen обмежується Dom0 bottleneck і меншою кількістю черг у класичній реалізації.

6. Тактико-технічні характеристики дискових підсистем

VirtIO-blk-mq + io_uring на хості дає відмінну multi-queue scalability і низький overhead. VMware PVSCSI оптимізований під enterprise storage (vSAN, FC, iSCSI) з динамічним coalescing запитів. Xen blkfront історично має меншу глибину черги і більший grant overhead, хоча persistent grants суттєво покращили ситуацію.

Табл. 3: Паспортні ТТХ дискових підсистем та обробки черг

Технічний параметр	VirtIO-Blk / SCSI	Xen Blkfront	VMware PVSCSI
Глибина черги (на LUN)	128–256 (до 1024/ctrl)	32–64 (типово)	32–254 (до 1024/ctrl)
Макс. IOPS на ВМ	1 000 000+ (io_uring)	100 000–300 000	1 000 000+ (vSAN/All-Flash)
Макс. дисків на ВМ	Багато / 256 на ctrl	До 256 пристроїв	До 4 ctrl × 64 = 256
Команди discard/TRIM	`discard` / UNMAP	barrier / flush / discard	UNMAP (SCSI)
Multi-Queue	Так (`virtio-blk-mq`)	Обмежено / ні	Так (з vSphere 7.0+)
Backend engine	io_uring / threads / native	blkback + grants	VMkernel + PSA

7. Високопродуктивні userspace-шляхи: DPDK, SPDK та прямий доступ до NVMe

Класичні паравіртуальні драйвери (VirtIO, Xen frontends, VMXNET3/PVSCSI) працюють у kernel space гостя і хоста. Під екстремальними навантаженнями (мільйони пакетів або IOPS на секунду) навіть оптимізований kernel path починає обмежувати продуктивність через context switch, interrupt handling і блокування. Саме тому з'явилися userspace-фреймворки, які повністю обходять ядро на datapath.

7.1. DPDK (Data Plane Development Kit)

DPDK — це набір бібліотек і драйверів для швидкої обробки мережевих пакетів у userspace. Основна ідея проста: замість interrupt-driven моделі ядро передає NIC безпосередньо додатку (через UIO/VFIO), а додаток працює в режимі постійного опитування (poll-mode driver, PMD).

Аргумент на користь polling: при packet rate понад ∼1–2 Mpps вартість обробки переривання стає вищою за вартість idle-опитування кільця. Тому DPDK свідомо «спалює» CPU-цикли на polling, отримуючи натомість передбачувану і дуже низьку latency.

Інтеграція з VirtIO реалізована через протокол `vhost-user`:

- На хості DPDK-додаток (OVS-DPDK, testpmd, власний VNF) виступає як vhost-user backend.
- У гості використовується virtio-pmd (також з DPDK).
- Virtqueues відображаються у спільну пам'ять; дані рухаються zero-copy.
- Kicks і completions можуть йти через eventfd / irqfd, мінімізуючи VM-Exits.

Результат: у конфігурації Physical-VM-Physical (PVP) з vhost-user + virtio-pmd можна досягти throughput, близького до wire-speed на 40/100 Gbit/s, з latency в одиниці мікросекунд. Класичний kernel vhost-net + virtio-net у тих самих умовах зазвичай програє в 2–4 рази.

Trade-off: DPDK вимагає pinned CPU cores, hugepages і відмови від стандартного мережевого стека ядра. Це прийнятно для NFV, телеком-шлюзів і high-frequency додатків, але ускладнює звичайне enterprise-адміністрування.

7.2. SPDK (Storage Performance Development Kit)

SPDK застосовує ту саму філософію до блокового I/O. NVMe-контролер відв'язується від kernel-драйвера (через uio/vfio) і передається userspace-додатку. SPDK реалізує повний NVMe-драйвер у userspace з poll-mode completion queues.

Чому це дає виграш?

1. Немає system call на кожен I/O.
2. Немає interrupt і пов'язаного з ним context switch.
3. Немає блокувань у block layer ядра.
4. Додаток може використовувати lockless черги і власний scheduler.

Типові цифри: один CPU core з SPDK здатний видавати понад 1–1.5 млн 4K random read IOPS на сучасному NVMe, тоді як kernel path (навіть з io_uring) зазвичай потребує більше ядер для тієї ж продуктивності.

У віртуалізації SPDK інтегрується двома основними способами:

- **vhost-user** (vhost-blk / vhost-scsi / vhost-nvme) — SPDK виступає backend для VirtIO-пристроїв гостя. QEMU лише налаштовує черги; datapath іде напряму між гостем і SPDK.
- **vfio-user** — більш сучасний протокол, який дозволяє SPDK емулювати повноцінний NVMe PCI-пристрій. Гостьова ОС (і навіть BIOS) бачить звичайний NVMe-контролер і використовує штатний драйвер. Це особливо важливо для Windows і для boot-from-NVMe сценаріїв, де VirtIO-драйвери можуть бути відсутні.

Аргумент: vfio-user знімає необхідність спеціальних драйверів у гості, зберігаючи при цьому майже ту саму продуктивність, що й vhost-user. Overhead порівняно з bare-metal NVMe становить одиниці мікросекунд.

7.3. Прямий доступ до NVMe та userspace-драйвери

Існує кілька рівнів «прямості» доступу до NVMe:

1. **PCI passthrough (VFIO / Xen PCI passthrough).** Пристрій повністю віддається одній ВМ. Гость отримує нативну продуктивність і повний контроль, але втрачається можливість live migration і sharing пристрою між кількома гостями. Використовується, коли потрібна абсолютна мінімальна latency.
2. **Userspace NVMe у самому гості.** Гость (або контейнер) може сам відв'язати NVMe від свого ядра і працювати через SPDK, NVMeDirect або аналогічні бібліотеки. Це дає максимальний контроль над I/O-плануванням всередині гостя.
3. **Командний passthrough.** Багато ОС (Linux з io_uring_cmd, FreeBSD, NetBSD) дозволяють надсилати сирі NVMe-команди через спеціальний інтерфейс, не проходячи повний block layer. NetBSD, зокрема, має розвинений nvme(4) драйвер з підтримкою per-CPU черг і command passthrough (nvmectl / відповідні ioctl). Це дозволяє будувати спеціалізовані userspace-інструменти без повного відв'язування пристрою.

NetBSD-контекст. NetBSD реалізує NVMe з акцентом на паралелізм: per-CPU I/O queue pairs, MSI-X, мінімальні блокування. Command passthrough дає можливість надсилати vendor-specific і administrative команди безпосередньо. Хоча повноцінного SPDK-подібного стеку в NetBSD немає, архітектура драйвера вже підготовлена до високопаралельної роботи і може слугувати основою для userspace-оптимізацій або для дослідження прямого доступу в BSD-середовищі.

7.4. Порівняння підходів

Висновок секції. DPDK і SPDK є логічним продовженням ідеї паравіртуалізації: якщо вже відмовляємося від повної емуляції, то наступний крок — прибрати і ядро з datapath. Це дає найкращі абсолютні цифри IOPS/latency, але вимагає спеціалізованого ПЗ і операційної моделі. Для більшості enterprise-сценаріїв достатньо добре налаштованого VirtIO/PVSCSI; для NFV, storage appliances і ultra-low-latency сервісів userspace-шлях стає практично обов'язковим.

Табл. 4: Порівняння kernel path, DPDK/SPDK та passthrough

Критерій	Kernel VirtIO/PV	DPDK / SPDK + vhost	PCI Passthrough
Latency (типова)	десятки µs	одиниці µs	близька до native
CPU efficiency	середня	висока (але pinned cores)	висока
Sharing пристрою	так	так (через backend)	ні
Live migration	так	обмежено / складно	зазвичай ні
Складність ops	низька	висока	середня
Підтримка Windows	через PV-драйвери	через vfio-user (NVMe)	нативна

8. Аналіз latency-шляхів і overhead

8.1. VirtIO latency path

Guest → descriptor write → avail.idx → kick (можливий VM-Exit) → vhost/QEMU processing → used.idx → interrupt/NAPI → guest completion. При vhost-user + polling VM-Exits майже зникають.

8.2. Xen latency path

Guest → ring request + grant ref → event channel → Dom0 backend map → native driver → unmap → response + event. Grant map/unmap і context switch у Dom0 додають latency, особливо під concurrency.

8.3. VMware latency path

Guest → shared ring descriptor → VMCALL / notification → VMkernel processing (EPT walk) → completion ring → virtual interrupt. Монолітний шлях без проміжного домену дає стабільну і передбачувану latency.

9. Безпека та ізоляція

- **Xen** — найсильніша модель ізоляції завдяки Grant Tables: Backend бачить лише явно надані сторінки. Stub domains додатково зменшують TCB.
- **VirtIO** — ізоляція залежить від гіпервізора (KVM + IOMMU / SEV тощо). Сам VirtIO не нав'язує grant-подібну модель.
- **VMware** — EPT + IOMMU + можливість DirectPath з правильним isolation. Сильна enterprise security model, але закрита.

10. Аналіз та обговорення

На основі отриманих паспортних TTX і аналізу механізмів можна зробити такі інженерні висновки:

1. **Продуктивність I/O:** Комбінація VirtIO з `vhost-user`/DPDK або `io_uring` демонструє найвищу ефективність і найменшу затримку завдяки відсутності проміжних доменів ізоляції та можливості повного userspace datapath. Multi-queue scalability (до 256 черг) є ключовою перевагою.
2. **Безпека:** Архітектура Xen із Grant Tables є найбільш захищеною від витоків пам'яті між гостями. Ціна — додаткові обчислювальні витрати на map/unmap і потенційний Dom0 bottleneck.

3. **Масштабованість підприємства:** VMware ESXi та контролери VMXNET3/PVSCSI забезпечують максимальну стабільність, багатий набір offloads (включаючи tunnel) і передбачуваність у гетерогенних корпоративних середовищах завдяки глибокій оптимізації монолітного VMkernel.
4. **Сучасні тенденції:** Усі три технології активно рухаються в бік SR-IOV / passthrough для ultra-low latency і в бік multi-queue + polling для high IOPS. VirtIO залишається найгнучкішим стандартом для open-source і cloud-native стеків.

11. Висновки

Порівняльний аналіз показав, що сучасні технології віртуалізації майже повністю нівелювали втрати продуктивності обчислень процесора (завдяки апаратній підтримці VT-x/AMD-V + EPT/NPT), перенісши конкуренцію на рівень периферійного введення-виведення.

- **VirtIO** домінує в сегменті відкритих, хмарних і high-performance userspace систем завдяки стандартизації, масштабованості черг і екосистемі DPDK/vhost.
- **VMware ESXi** залишається еталоном для Enterprise-сегменту з вимогами до стабільності, offloads і інтеграції зі storage/network fabric.
- **Xen** зберігає позиції в архітектурах із підвищеними вимогами до ізоляції та безпеки (зокрема в системах з мінімальним TCB).

Подальші дослідження доцільно зосередити на quantitative benchmark'ах (fio, pktgen, DPDK testpmd) під ідентичним hardware з вимірюванням tail latency (p99/p999) та CPU cycles per I/O, а також на впливі нових апаратних можливостей (IOMMU, SmartNICs, DPUs).