Avaliação Comparativa de Desempenho entre Contêineres e Máquinas Virtuais Baseadas em KVM

Davi Kniest[1], Artur Z. Marcon[1], Julio A. de C. Borges[2], A. Sacilotto[1], Miguel G. Xavier[1], Tiago C. Ferreto [1], Cesar A. F. De Rose[1]

[1]Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – 90.619-900 – Porto Alegre – RS – Brasil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{andre.santos01, artur.marcon, davi.silva004}@edu.pucrs.br

jacborges@inf.ufrgs.br

{cesar.derose, miguel.xavier, tiago.ferreto}@pucrs.br

Abstract. *Virtualization technologies have evolved significantly over the last decade. Past studies have shown that containers achieve better performance than virtual machines. This work addresses questions raised over the years, empirically investigating the efficacy of these platforms and contrasting operating system solutions and hypervisors executed on a specific hardware platform. Docker, Podman, LXC, and KVM were evaluated in comparison to native performance, measuring CPU, memory, disk, and network under various workloads. The results show that containers maintain near-native performance, while the hypervisor achieves competitive performance through workload-specific adjustments.*

Resumo. *As tecnologias de virtualização evoluíram significativamente na última década. Estudos anteriores mostram que contêineres apresentam desempenho superior ao de máquinas virtuais. Este trabalho responde a questões levantadas ao longo de anos, investigando empiricamente a eficácia dessas plataformas e contrastando soluções de sistemas operacionais e hypervisors executadas sobre uma plataforma de hardware específica. Docker, Podman, LXC e KVM foram avaliados em comparação com o desempenho nativo, medindo CPU, memória, disco e rede sob diversas cargas. Os resultados mostram que os contêineres mantêm desempenho quase nativo, enquanto o hypervisor alcança desempenho competitivo por meio de ajustes específicos por carga de trabalho.*

1. Introdução

A virtualização consolidou-se como um pilar essencial das infraestruturas modernas de computação em nuvem, viabilizando o isolamento de recursos, a consolidação de servidores e o provisionamento dinâmico e elástico de aplicações [Deng et al. 2024, Pahl et al. 2019]. Nesse cenário, a tecnologia de contêineres tem se destacado, oferecendo uma alternativa mais leve e com consumo de recursos de hardware mais eficiente em comparação com as máquinas virtuais tradicionais [Pahl et al. 2019].

No âmbito da computação de alto desempenho (HPC), diversos estudos investigam se as tecnologias de virtualização conseguem aliar a flexibilidade de implantação à preservação de um desempenho próximo ao do ambiente nativo (bare-metal) [Xavier et al. 2013, Kurtzer et al. 2017, Torrez et al. 2019]. Demonstrou-se que a virtualização baseada em contêineres apresenta *overhead* expressivamente menor em comparação às soluções de *hypervisor* para diferentes cargas de HPC, ao custo de um menor isolamento de recursos [Xavier et al. 2013]. Avaliações subsequentes estenderam a comparação a ambientes de nuvem [Felter et al. 2015, Morabito et al. 2015, Chae et al. 2019], consolidando o entendimento de que contêineres operam próximos ao hardware nativo.

Contudo, o ecossistema passou por transformações significativas na última década: a consolidação do padrão *Open Container Initiative* (OCI) e de múltiplos *runtimes* (Docker, Podman, LXC), a execução *rootless*, a hierarquia unificada de cgroups v2, os avanços na paravirtualização de E/S e as extensões de virtualização assistida por hardware [Djordjevic et al. 2022, Pahl et al. 2019]. Consequentemente, as avaliações de desempenho exigem atualizações contínuas para refletir versões e configurações atuais do software de virtualização, visto que o *overhead* atual apresenta variações significativas que dependem da carga, da parametrização do sistema e da implementação específica de cada *runtime* [Abraham et al. 2020, Giallorenzo et al. 2021].

Este trabalho propõe uma nova análise do desempenho de virtualização por meio de um estudo comparativo entre KVM, Docker, Podman e LXC em relação ao ambiente nativo, com um conjunto de benchmarks que abrange processamento, memória, armazenamento, rede e aplicações paralelas. As contribuições principais são: (i) uma atualização empírica dos *trade-offs* entre contêineres e VMs em kernels e runtimes atuais; (ii) a decomposição do *overhead* do *hypervisor* em custo arquitetural e custo de configuração, por meio de múltiplos ajustes. As Seções 2 a 6 apresentam, respectivamente, os fundamentos das tecnologias avaliadas, a metodologia dos experimentos, os resultados, os trabalhos relacionados e as conclusões.

2. Fundamentação Teórica

Esta seção apresenta os fundamentos arquiteturais e as distinções entre as tecnologias de virtualização avaliadas neste estudo. A descrição das implementações específicas abordadas nesta pesquisa é apresentada nas respectivas seções, juntamente com os conceitos de abstração de hardware e de nível de sistema operacional, de forma integrada.

2.1. Kernel-based Virtual Machine (KVM)

O Kernel-based Virtual Machine (KVM) [Linux KVM Project 2014] é um *hypervisor* integrado ao kernel do Linux. Esta tecnologia introduz uma camada de software intermediária, o Monitor de Máquina Virtual (Virtual Machine Monitor, VMM), entre o hardware físico e os sistemas operacionais convidados (*guests*).

Ao utilizar o KVM, o sistema operacional hospedeiro passa a operar, na prática, como um *hypervisor* do tipo 1 (bare-metal), uma vez que o módulo executa diretamente no hardware, com as extensões de virtualização. O *hypervisor* atua como árbitro centralizador, responsável por interceptar chamadas de sistema privilegiadas, escalonar as CPUs virtuais (vCPUs) entre os núcleos físicos disponíveis e gerenciar o acesso seguro à memória e aos periféricos de Entrada/Saída (E/S). Para reduzir a latência de emulação, o KVM utiliza extensões de virtualização assistida por hardware (como Intel VT-x e AMD-V), delegando transições de estado críticas diretamente à CPU. O gerenciamento de memória depende de tabelas de páginas estendidas (Extended Page Tables ou EPT) para traduções aceleradas por hardware. Adicionalmente, utiliza o framework `virtio` [Russell 2008] para a paravirtualização de dispositivos de rede e de bloco. Essa arquitetura, embora garanta o particionamento de recursos em Máquinas Virtuais (VMs) estritamente isoladas, introduz *overheads* estruturais cuja magnitude é quantificada na Seção 4.

2.2. Linux Containers (LXC)

O LXC representa uma das abstrações primárias da virtualização em nível de sistema operacional. Em contraste com o KVM, essa abordagem não introduz uma camada de virtualização de hardware para cada instância. Em vez disso, o LXC utiliza mecanismos de contenção fornecidos pelo kernel Linux, como *namespaces* e *cgroups*, permitindo que múltiplos ambientes isolados compartilhem o kernel do sistema hospedeiro [Linux Containers Org 2025].

Para alcançar o isolamento de recursos sem recorrer à emulação, o LXC depende estritamente de duas primitivas fundamentais do kernel Linux: os `namespaces` e os `cgroups` (*control groups*). Os `namespaces` particionam a visibilidade dos recursos globais do sistema, garantindo que um processo virtualizado possua uma visão independente (identificadores de processos, pilhas de rede, comunicação entre processos e pontos de montagem) [Biederman 2006]. Simultaneamente, os `cgroups` atuam como o subsistema de gerenciamento de recursos, impondo limites superiores e contabilizando ciclos de CPU, consumo de memória e largura de banda de E/S [Heo 2015].

O LXC utiliza esses mecanismos não para encapsular um único serviço, mas para fornecer *contêineres de sistema* completos, capazes de executar um sistema de inicialização inteiro (`systemd` ou `sysvinit`) com múltiplos processos, sendo adequado para a consolidação de servidores clássicos ou de instâncias de longa duração, nas quais se exige controle total do ambiente operacional, sem as penalidades de tradução do KVM [Linux Containers Org 2025].

2.3. Docker

O Docker expande as primitivas de virtualização em nível de sistema operacional introduzidas pelos `namespaces` e `cgroups`, refinando a conteinerização para focar no empacotamento e no isolamento de aplicações individuais e de suas dependências (*application containers*). Diferenciando-se da abordagem de sistema completo do LXC, o Docker introduziu um modelo de imagens em camadas baseado em sistemas de arquivos de união (UnionFS) e descrito declarativamente em `Dockerfiles`, o que viabiliza a reprodução imutável e determinística de ambientes [Turnbull 2014].

Nesta arquitetura, cada contêiner opera como um processo no espaço de usuário (*user-space*), intermediado por um *daemon* central de longa execução (`dockerd`), que delega a criação dos contêineres ao `containerd` e, na camada inferior, ao runtime

OCI `runc` [Open Container Initiative 2023]. A plataforma consolidou a distribuição de imagens por repositórios (Docker Hub) e a orquestração local com `docker-compose`. Graças a essas abstrações voltadas ao ciclo de desenvolvimento de software, o Docker padronizou a virtualização baseada em contêineres na indústria de tecnologia [Felter et al. 2015, Bernstein 2014].

2.4. Podman

O Podman é uma tecnologia de containerização desenvolvida pela Red Hat, com foco na correção de gargalos de segurança sistêmica e na eliminação de pontos únicos de falha no modelo adotado pelo Docker. A principal divergência arquitetural do Podman reside em seu modelo daemonless (sem processos persistentes em segundo plano).

Sem a necessidade de um serviço central (como o `dockerd`), cada contêiner no Podman é executado diretamente como um processo filho do processo do usuário que o invocou. Isso viabiliza a execução de contêineres sem privilégios de superusuário (rootless), assegurando conformidade estrita com o princípio do menor privilégio [Red Hat, Inc. 2023]. A ferramenta adere nativamente à especificação de tempo de execução da Open Container Initiative (OCI), utilizando o `runc` ou o `crun` como motores subjacentes [Open Container Initiative 2023, Scrivano 2023]. O design descentralizado do Podman também permite uma integração profunda com o `systemd` do hospedeiro, o que facilita o gerenciamento nativo do ciclo de vida dos serviços em ambientes multiusuário avançados [Red Hat, Inc. 2026].

3. Metodologia

Esta sessão descreve a metodologia utilizada para os experimentos assim como o ambiente de execução. Os experimentos foram executados em um servidor seguindo a especificação da Tabela 1. As versões avaliadas foram: Docker 29.1.3, Podman 4.9.3, LXC 5.0.3 LTS, `libvirt` 10.0.0 e QEMU 8.2.2 com `virtio-blk` e `virtio-net` (`vhost-net`). A avaliação foi conduzida em uma única plataforma física, cujas características delimitam o escopo dos resultados apresentados com relação ao hardware.

Modelo	Dell® PowerEdge R720
CPU	2 soquetes 16C (32T) Intel(R) Xeon(R) E5-2650 v2
RAM	128GB (8x 16GB - DDR3 Registered ECC - 1600MT/s)
Disco	HDD 600GB (2x 300GB - 15K RPM - SAS 6Gbps) SSD 960GB (2x 480GB - 12 Gbps)
Rede	NetXtreme BCM5720 Gigabit Ethernet PCIe
SO	Ubuntu® Server, 24.04 LTS
Kernel	Linux 6.8.0

Tabela 1. *Hardware* utilizado com modelos e especificações.

As tecnologias de virtualização foram avaliadas com as suas configurações padrão, com exceção do KVM, o qual é conhecido na literatura por precisar de alterações para funcionar de forma satisfatória em ambientes de HPC. Dessa forma, para avaliar o impacto das configurações sobre a sobrecarga do *hypervisor*, duas configurações foram utilizadas para o KVM: uma próxima ao padrão e outra otimizada para a arquitetura *Non-Uniform Memory Access* (NUMA), na qual o tempo de acesso à memória depende de sua localização, sendo menor para acessos locais e maior para acessos remotos. As configurações são descritas a seguir:

i. *KVM-std*, com a configuração padrão do `libvirt` e as alterações `<cache='none'>` e `<io='native'>` para remover o cache do *host* e delegar E/S para bibliotecas nativas do *host*;
ii. *KVM-opt* com as mesmas mudanças da configuração anterior e otimizações para reduzir o *overhead* já conhecido. Isso foi possível por meio das opções: `<numatune mode='strict'>` que fixa a memória em um nó NUMA, `<cpu mode='host-passthrough'>` que expõe diretamente os recursos da CPU do *host* e o uso de *hugepages* via `<memoryBacking>` para habilitar páginas de memória maiores.

A metodologia dos testes foi estruturada a partir de dois conjuntos principais de scripts: um responsável pela preparação e orquestração das infraestruturas e outro pela execução dos benchmarks. Os scripts de infraestrutura controlavam as diferentes tecnologias de virtualização disponíveis no servidor, garantindo que apenas a infraestrutura correspondente à bateria de testes em execução permanecesse ativa, de modo a reduzir possíveis interferências entre os ambientes avaliados. Além disso, entre as execuções dos benchmarks, eram realizadas rotinas de limpeza de caches e de resíduos deixados por execuções anteriores, buscando manter condições experimentais consistentes. Os scripts de benchmark, por sua vez, eram responsáveis por selecionar e iniciar as ferramentas de avaliação definidas para cada experimento, configurando suas execuções de acordo com os parâmetros específicos exigidos por cada benchmark e cenário experimental. Cada combinação de benchmark e ambiente avaliado foi executada 10 vezes, e os resultados apresentados correspondem à média dessas repetições.

4. Experimentos e Resultados

Esta seção detalha a avaliação empírica do desempenho proporcionado por tecnologias de virtualização baseadas em contêineres e em *hypervisors*. A bateria de experimentos foi conduzida por meio de conjuntos de benchmark selecionados para isolar e estressar subsistemas específicos, culminando na avaliação de aplicações paralelas reais (HPC). O objetivo é correlacionar as métricas de desempenho com as diferenças arquiteturais subjacentes de cada plataforma.

4.1. Desempenho de CPU

Para avaliar a capacidade computacional bruta e a eficiência no escalonamento de instruções, o benchmark LINPACK [Dongarra et al. 2003] foi utilizado em modo *single-thread*. Este teste quantifica o desempenho de ponto flutuante por meio da resolução de sistemas lineares densos, o padrão de fato para cargas de trabalho limitadas pela CPU (CPU-bound).

Em todos os cenários avaliados, as plataformas de virtualização baseadas em contêineres (Docker, Podman e LXC) atingiram um desempenho computacional virtualmente indistinguível da execução nativa, operando na faixa de 1830 a 1900 Mflops, com variação dentro do intervalo de confiança da linha de base *bare-metal* ($\sim$1800 Mflops) (Figura 1), correspondendo à literatura [Felter et al. 2015]. Esse comportamento é justificado pela arquitetura de contêineres: como os processos compartilham o kernel do *host*, eles são gerenciados diretamente pelo escalonador do sistema operacional (por exemplo, o *Completely Fair Scheduler*). Isso permite a execução direta no hardware e a exploração integral de conjuntos de instruções de vetorização avançados, como AVX e FMA, sem qualquer penalidade de tradução. O leve ganho pontual frente ao *bare-metal* reforça a eficácia dos *cgroups* em mitigar ruídos de *background* do sistema.

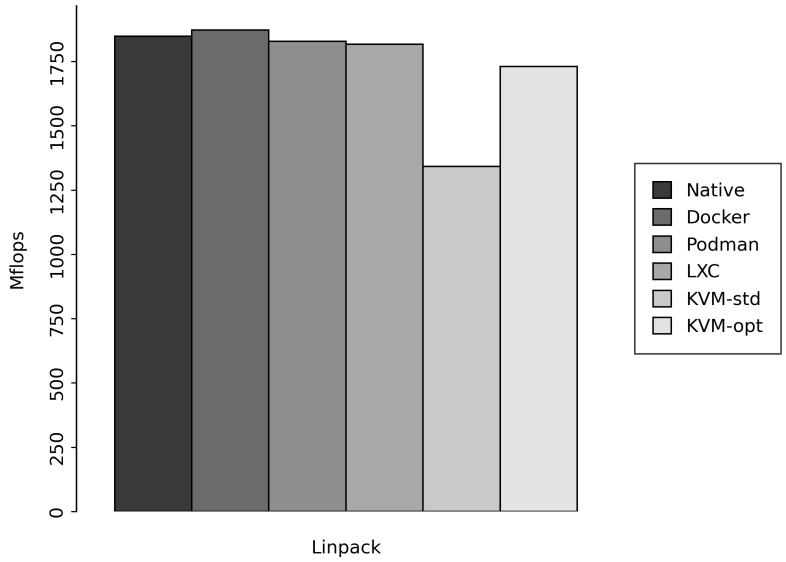

Figura 1. Desempenho de CPU (LINPACK): MFLOPS por ambiente

A virtualização baseada em *hypervisor*, por sua vez, apresentou um comportamento fortemente dependente de configuração. Na configuração padrão, o *KVM-std* registrou um *overhead* de aproximadamente 25% na taxa de processamento (atingindo apenas $\sim$1340 Mflops). Como ambas as VMs foram instanciadas com acesso integral às instruções físicas da CPU (*host-passthrough*), esta degradação decorre primariamente da ausência de alinhamento com a topologia do hardware e da falta de afinidade do escalonador: sem o mapeamento NUMA correto, as *threads* do SO convidado são migradas livremente entre os dois soquetes físicos do servidor hospedeiro. Isso gera tráfego de memória cruzado, no qual a CPU aguarda dados do nó vizinho, causando interrupções no processamento vetorial (além do custo natural das *VM-exits* e *VM-entries*). Já a configuração *KVM-opt*, operando com a topologia NUMA 1:1 espelhada e as vCPUs ancoradas aos núcleos físicos (*pinning*), mitigou a latência de memória e reduziu o *overhead* para aproximadamente 4% ($\sim$1730 Mflops). Isso aproximou virtualmente a VM do desempenho nativo, corroborando a observação de [Felter et al. 2015] de que grande parte do custo de CPU historicamente atribuído ao *hypervisor* é, na verdade, decorrente de uma configuração inadequada.

4.2. Desempenho de Memória

A vazão do subsistema de memória foi avaliada por meio do benchmark STREAM [McCalpin 1995]. A ferramenta executa operações vetoriais contínuas (Copy, Scale, Add, Triad) em arranjos de dados que excedem intencionalmente a capacidade das caches do processador, medindo a largura de banda de memória efetiva de forma realista.

Os motores de contêineres (Docker, Podman e LXC) mantiveram paridade com o hardware nativo em todas as operações, limitando a variação a uma margem desprezível (Figura 2). Essa eficiência decorre do mapeamento direto de memória: os contêineres utilizam as mesmas tabelas de páginas e a mesma Unidade de Gerenciamento de Memória (MMU) do *host*, operando com o mesmo custo de tradução de endereços virtuais para físicos que um processo nativo [Morabito et al. 2015].

A configuração *KVM-std* sofreu uma redução de 11,6% a 21,0% na largura de

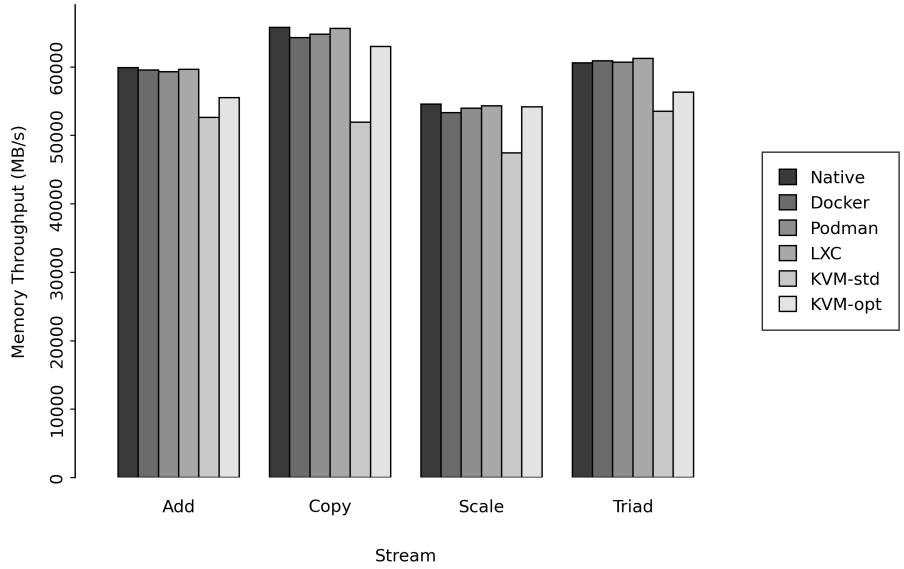

Figura 2. Largura de banda de memória efetiva (STREAM) por operação e ambiente

banda. Parte substancial dessa penalidade decorre da alocação de memória insensível à topologia NUMA: quando os acessos vetoriais do STREAM extrapolam a capacidade do *Translation Lookaside Buffer* (TLB), o hardware é forçado a realizar o *page walk* bidimensional exigido pelas *Extended Page Tables* (EPT) [Bhargava et al. 2008, Merrifield and Taheri 2016], cujo custo de latência é amplificado quando as páginas físicas residem em um nó NUMA remoto (obrigando o tráfego a atravessar o barramento de interconexão entre *sockets*). De forma consistente com essa interpretação, a configuração *KVM-opt* recuperou a largura de banda para níveis muito mais próximos aos do ambiente nativo (restringindo a defasagem a apenas 0,7% a 7,2%, conforme a operação), demonstrando que o gargalo dominante de memória neste cenário não é puramente estrutural ao modelo da máquina virtual, mas sim intrinsecamente ligado à ausência de afinidade NUMA [Felter et al. 2015].

4.3. Desempenho de E/S (I/O)

A análise da pilha de armazenamento foi conduzida utilizando o IOzone [Norcott and Capps 2015], mantendo a comparabilidade histórica com a metodologia de trabalhos anteriores [Xavier et al. 2013]. Para estes experimentos iniciais, a infraestrutura de armazenamento operou no disco principal e sob o sistema de arquivos padrão do hospedeiro (modo sem arranjo RAID isolado). O acesso aos arquivos de teste ocorreu de forma divergente entre as plataformas: os contêineres utilizaram mapeamentos diretos de diretórios (*bind mounts*), enquanto as máquinas virtuais sob o *hypervisor* KVM operaram internamente sobre discos virtuais baseados em arquivos de imagem (formato .raw).

Os resultados para as operações estritas de *initial write*, *rewrite*, *read* e *reread* (Figura 3) demonstraram que não houve sobrecarga de virtualização mensurável em nenhum dos ambientes analisados, com as barras de KVM e contêineres igualando-se estatisticamente ao desempenho nativo. Isso ocorre devido à configuração da ferramenta para utilizar a *flag* de acesso direto (O_DIRECT) [Felter et al. 2015]. Ao forçar o I/O direto, o subsistema de cache do sistema operacional (page cache) é ignorado e as requisições fluem diretamente para o hardware. Neste cenário, a latência e a vazão da operação são

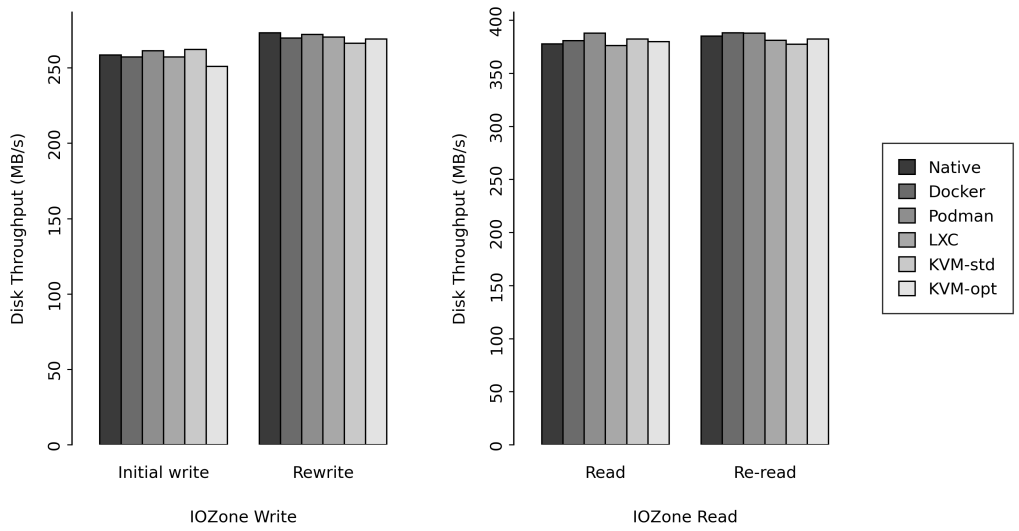

Figura 3. Vazão de pico de E/S de arquivo (IOzone) por operação e por ambiente.

estranguladas pelas limitações mecânicas ou eletrônicas do disco físico. Como o hardware físico de armazenamento é muito mais lento que o processamento do *hypervisor*, o *overhead* computacional de tradução de chamadas do KVM ou do VFS nos contêineres é mascarado pelo tempo de resposta do disco.

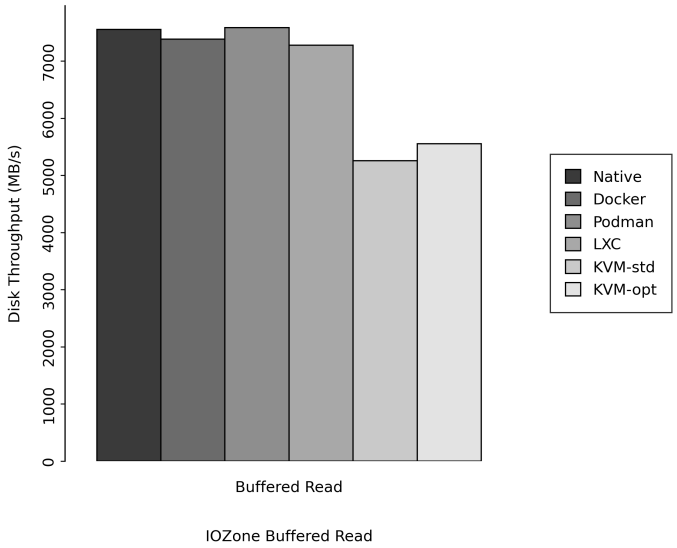

Figura 4. Vazão de pico de E/S de arquivo (IOzone) na operação de leitura com *buffer*, por ambiente.

O real impacto das diferentes arquiteturas de virtualização só é revelado quando a leitura em *buffer* (*buffered read* / *fread*) é avaliada (Figura 4). Ao remover a instrução de I/O direto, os dados passam a ser requisitados diretamente do cache em memória RAM. Ao eliminar o gargalo do disco físico da equação, o teste passa a mensurar essencialmente a vazão de memória e o custo das transições de contexto do kernel. Neste cenário, as máquinas virtuais (KVM-std e KVM-opt) apresentam queda no desempenho. Essa queda de *throughput* é causada pelo fenômeno de "cache duplo" e redundância de cópias de memória exigidas pelo *hypervisor*: os dados armazenados no

buffer do hospedeiro precisam ser iterativamente copiados para o processo de emulação (QEMU) e, então, mapeados para o espaço de memória do sistema operacional convidado [Sharma and Kulkarni 2012]. Em contraste, os contêineres preservam alto desempenho, por contarem com o compartilhamento direto do kernel hospedeiro, evitando cópias excessivas de memória e emulação de hardware.

4.4. Desempenho de Rede

O desempenho da pilha de rede entre nós foi avaliado por meio do NetPIPE [Snell et al. 1996]. A ferramenta executa testes de comunicação ponto a ponto (*ping-pong*) para quantificar a latência em mensagens pequenas e a vazão em mensagens grandes. Os experimentos ocorreram entre duas máquinas físicas interconectadas por uma rede Gigabit Ethernet.

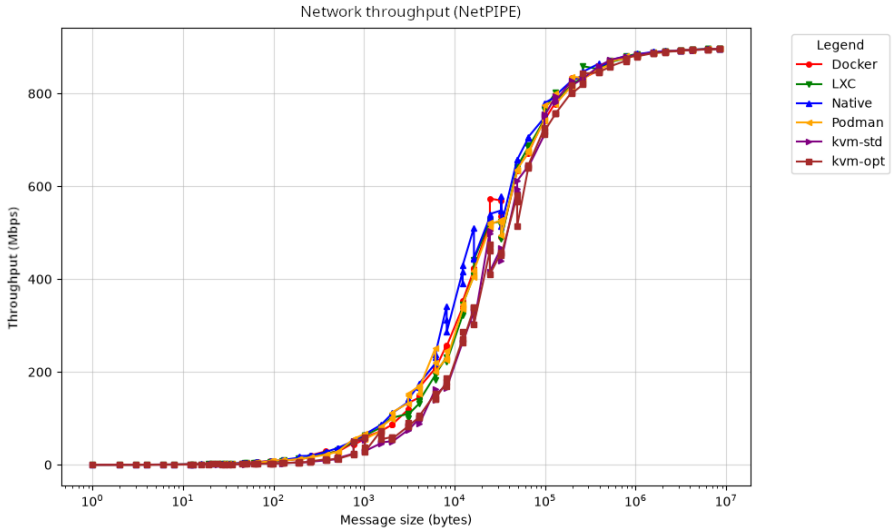

Figura 5. Vazão alcançada (NetPIPE) por tamanho de mensagem e por ambiente

Para mensagens grandes ($\sim$8 MB), todas as plataformas saturaram a interface Gigabit física (Figura 5). Tanto contêineres quanto máquinas virtuais (*KVM-std* e *KVM-opt*) atingiram vazão de pico em torno de 896 Mbps, equiparando-se estatisticamente ao ambiente nativo. O *overhead* de virtualização, como as transições de contexto e os saltos por pontes virtuais (*bridges*), é amortizado na transferência de grandes volumes de dados, restringindo o desempenho à capacidade de transmissão do enlace físico [Felter et al. 2015]. As diferenças estruturais entre as tecnologias ficam evidentes na latência para mensagens de 1 byte. O ambiente nativo obteve uma latência de $\sim$75 μs. Os contêineres (Docker, Podman e LXC) apresentaram *overhead* contido, registrando entre $\sim$98 μs e $\sim$112 μs. O custo adicional decorre do roteamento da rede sobreposta, em que os pacotes percorrem pares de interfaces virtuais (*veth pair*) e pontes de software no hospedeiro (como `docker0` ou `cni0`) antes de alcançar a interface física.

Nas máquinas virtuais, o tempo de resposta operou na faixa de 312 a 316 μs, valor superior a quatro vezes a latência nativa. Diferentemente do comportamento nos subsistemas de CPU e memória, o ajuste de topologia na configuração *KVM-opt* não mitigou o *overhead*. Essa latência adicional é inerente à complexidade do caminho de dados no modelo de virtualização de E/S do *hypervisor*. O fluxo de cada pacote exige o mapeamento e notificação de anéis de memória compartilhada (*virtqueues*) [Russell 2008] pelo

driver `virtio-net`, impondo sucessivas trocas de contexto (*VM-exits* e *VM-entries*) até que os dados sejam processados pelo módulo `vhost-net` no *kernel* do hospedeiro [Hwang et al. 2013]. Em cargas baseadas na emissão granular e sequencial de pacotes, essa arquitetura se torna o gargalo dominante.

4.5. Aplicações Científicas Paralelas (HPC)

Para complementar a avaliação de desempenho computacional, a coleção de benchmarks NAS Parallel Benchmarks (NPB) [Bailey et al. 1991] foi utilizada em implementação OpenMP (classe C), que avalia o impacto sistêmico da virtualização sobre cargas científicas multithread, abrangendo desde kernels computacionais estritos (EP, IS, CG, MG, FT) até pseudoaplicações que simulam fluidodinâmica (BT, LU).

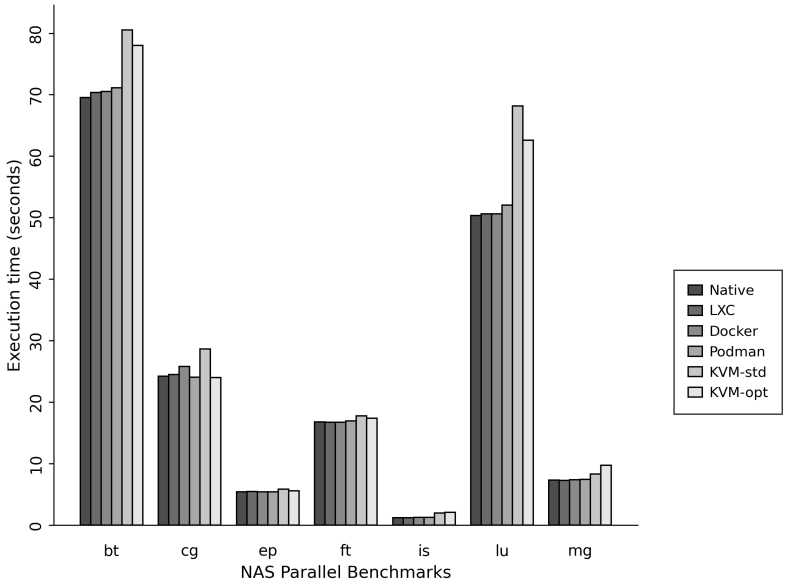

Figura 6. Tempo de execução (NPB) por kernels e por ambiente.

Os resultados (Figura 6) sintetizam as limitações expostas nos testes de subsistemas. O kernel *Embarrassingly Parallel* (EP) comprovou sua eficiência computacional, com *overhead* desprezível em todos os ambientes. Os contêineres igualaram o tempo de execução nativo em todas as pseudoaplicações. A VM padrão (KVM-std), em contrapartida, apresentou aumento no tempo de execução de até 35,4% nas cargas sensíveis à hierarquia de memória, como o LU (Lower-Upper Gauss-Seidel solver), e de 15,8% no BT (Block Tri-diagonal solver). Esse comportamento é coerente com a penalidade de latência e limitação de largura de banda observadas no STREAM, causadas pela migração indiscriminada de vCPUs entre os *sockets* físicos da máquina [Rao et al. 2013]. A configuração otimizada (KVM-opt) conseguiu recuperar parte expressiva dessa perda por meio do *vCPU pinning* [Cheng et al. 2013], reduzindo a diferença de tempo para 24,3% (no LU) e 12,2% (no BT), reforçando o papel dominante do respeito à topologia NUMA para atenuar gargalos de memória. Cabe destacar que o presente estudo foi delimitado ao escopo arquitetural *single-node* (visando avaliar os custos intra-nó), remetendo a avaliação de cargas distribuídas *multi-node* (como implementações baseadas em MPI em rede virtualizada) para trabalhos futuros.

5. Trabalhos Relacionados

Ao considerar HPC, virtualização e interferência no desempenho, [Xavier et al. 2013] avaliaram o compromisso entre desempenho bruto e isolamento nas implementações iniciais de virtualização em nível de sistema operacional (Linux-VServer, OpenVZ e LXC), contrastando-as com o Xen, *hypervisor* predominante à época, por meio de benchmarks de subsistemas e do conjunto NAS Parallel Benchmarks (NPB) em OpenMP e MPI. Os contêineres obtiveram vazão quase nativa em processamento, memória e disco, com a divergência mais expressiva na rede: o *hypervisor* sofreu degradação de até 41% na vazão, penalizando fortemente a escalabilidade multinodo do NPB, enquanto os contêineres mantiveram a eficiência de paralelização.

O mesmo estudo revelou problemas de isolamento: com a Isolation Benchmark Suite (IBS) [Matthews et al. 2007], os autores mostraram que, à exceção da CPU, a saturação de memória, disco ou rede por um vizinho colocalizado gerava severa degradação nos demais processos, pois o kernel compartilhado atuava como gargalo sistêmico. Essa constatação embasou o entendimento clássico de que contêineres entregam desempenho irrestrito com isolamento permeável, enquanto *hypervisors* garantem isolamento estrito à custa de desempenho.

Expandindo esta análise para além dos arranjos HPC restritos, [Sharma et al. 2016] investigaram a dinâmica entre a virtualização em nível de sistema operacional e a virtualização em nível de hardware em escala de data center. A pesquisa abrangeu métricas de desempenho de base, a tolerância ao superprovisionamento de recursos e o impacto arquitetural imposto por orquestradores corporativos, como OpenStack, Kubernetes e Docker Swarm. Os resultados corroboraram a tese de que contêineres podem ter desempenho quase bare-metal em rotinas limitadas por CPU e memória, porém evidenciaram que a arquitetura de kernel compartilhado agrava o impacto da contenção em cenários de locação múltipla (multi-tenant) ou sob o comportamento de processos adversários. Como solução, o estudo defendeu a adoção de estratégias híbridas de implantação, encapsulando contêineres em máquinas virtuais leves.

Mais recentemente, a adoção dessas plataformas para o processamento de dados foi analisada por [Zhang et al. 2018], que examinaram a escalabilidade de frameworks de big data (notadamente o Apache Spark) operando em contêineres e em VMs. Por meio de cargas de trabalho como K-Means, Regressão Logística, PageRank e junções SQL distribuídas, o estudo quantificou que ambientes conteinerizados reduzem a sobrecarga de inicialização e elevam a densidade de utilização do cluster. A supressão da camada do *hypervisor* resultou em tempos de conclusão de tarefas significativamente mais curtos e em uma utilização mais eficiente dos recursos em paralelo. Os autores alertaram, no entanto, que a orquestração de contêineres em domínios altamente distribuídos impõe desafios multiusuários existentes em relação à migração de estado, reiterando a necessidade de refinamentos estruturais contínuos para garantir o isolamento sem onerar a flexibilidade operacional.

No eixo das comparações diretas entre contêineres e VMs, [Felter et al. 2015] demonstraram, ainda em 2015, que grande parte dos custos de CPU e de memória do KVM desaparece com a fixação de vCPUs e a exposição da topologia de cache, deslocando as penalidades remanescentes para os caminhos de E/S e de rede. Avaliações posteriores em nuvem [Morabito et al. 2015, Chae et al. 2019] e contra bare-metal [Giallorenzo et al. 2021] corroboraram esse quadro, cada qual reportando variações

dependentes da carga e da parametrização. No domínio HPC, [Torrez et al. 2019] mostraram que os runtimes de contêineres voltados à supercomputação impõem impacto mínimo ou nulo no desempenho, e [Abraham et al. 2020] analisaram os efeitos das camadas de armazenamento de contêineres sobre cargas científicas intensivas em E/S.

Este trabalho distingue-se dessas iniciativas em dois principais aspectos: primeiro, atualiza a comparação sistemática para runtimes contemporâneos (incluindo o Podman daemonless/rootless) sob kernels e *cgroups v2* atuais; segundo, decompõe explicitamente o *overhead* do *hypervisor* em custo arquitetural e custo de configuração por meio de ajustes ao KVM, dimensão ausente das comparações anteriores, que tratam a VM como configuração única.

6. Conclusão e Trabalhos Futuros

Os resultados demonstram que runtimes de contêineres (Docker, Podman e LXC) atingem paridade com o desempenho bare-metal nos subsistemas avaliados e na execução de aplicações científicas HPC. Para cargas estritamente limitadas por E/S e latência de rede, ambas as arquiteturas de virtualização impõem penalidades específicas que exigem parametrização.

A degradação de desempenho do KVM em cargas *multithread* decorre primariamente da ausência de alinhamento com a topologia de hardware. A aplicação de afinidade NUMA e a ancoragem de vCPUs aos núcleos físicos (KVM-opt) mitigam o tráfego de memória cruzado e aproximam a vazão de processamento da VM aos níveis nativos. Contudo, a arquitetura do *hypervisor* mantém um overhead mensurável e inerente em operações de E/S com *buffer* e no processamento de pacotes de rede granulares, em função do custo das transições de contexto (VM-exits/VM-entries).

Nos contêineres, o compartilhamento direto do kernel suprime os gargalos de E/S em disco e memória observados nas VMs. No entanto, o roteamento de pacotes através de *namespaces* e pontes virtuais introduz latência de rede adicional frente ao acesso físico nativo. Este fator estabelece um *trade-off* técnico direto entre o nível de isolamento de rede implementado e a capacidade de sincronização entre nós da aplicação.

Embora a virtualização seja historicamente associada a penalidades de desempenho restritivas, os resultados deste trabalho estabelecem uma perspectiva otimista ao demonstrar que o ajuste arquitetural adequado reduz essa sobrecarga a níveis marginais. Constata-se que, mediante a correta parametrização do ambiente, as cargas de trabalho operam com eficiência próxima à nativa, superando a noção de um custo computacional inevitável. Uma vez que o desempenho bruto deixa de atuar como o principal fator limitante dessas infraestruturas, a eficácia e a segurança dos mecanismos de isolamento assumem o papel de requisito mais crítico para a viabilidade e a consolidação dessas plataformas.

Os experimentos foram realizados em um único servidor. Portanto, os resultados refletem a configuração de hardware investigada e não são necessariamente generalizáveis para plataformas com diferentes características microarquiteturais e topologias NUMA.

Trabalhos futuros focarão na avaliação do isolamento de desempenho em cenários de concorrência (multi-tenant). O objetivo será quantificar a degradação causada por processos adversários e verificar se a contenção de recursos imposta pelos *cgroups* nos contêineres equipara-se à eficácia do isolamento em nível de hardware proporcionado pelo *hypervisor*.

Referências

Abraham, S., Paul, A. K., Khan, R. I. S., and Butt, A. R. (2020). On the use of containers in high performance computing environments. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 284–293. IEEE.

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. (1991). The NAS parallel benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73.

Bernstein, D. (2014). Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Bhargava, R., Serebrin, B., Spadini, F., and Manne, S. (2008). Accelerating two-dimensional page walks for virtualized systems. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 26–35. ACM.

Biederman, E. W. (2006). Multiple instances of the global Linux namespaces. In *Proceedings of the Linux Symposium*, volume 1, Ottawa, Canada.

Chae, M., Lee, H., and Lee, K. (2019). A performance comparison of Linux containers and virtual machines using Docker and KVM. *Cluster Computing*, 22(Suppl 1):1765–1775.

Cheng, Y., Chen, W., Chen, X., Xu, B., and Zhang, S. (2013). A user-level numa-aware scheduler for optimizing virtual machine performance. In *International Workshop on Advanced Parallel Processing Technologies*, pages 32–46. Springer.

Deng, S., Zhao, H., Huang, B., Zhang, C., Chen, F., Deng, Y., Yin, J., Dustdar, S., and Zomaya, A. Y. (2024). Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE*, 112(1):12–46.

Djordjevic, B., Timcenko, V., Lazic, M., and Davidovic, N. (2022). Performance comparison of Docker and Podman container-based virtualization. In *2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6. IEEE.

Dongarra, J. J., Luszczek, P., and Petitet, A. (2003). The LINPACK benchmark: Past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172. IEEE.

Giallorenzo, S., Mauro, J., Poulsen, M. G., and Siroky, F. (2021). Virtualization costs: Benchmarking containers and virtual machines against bare-metal. *SN Computer Science*, 2(5):404.

Heo, T. (2015). Control group v2. Documentação do kernel Linux, `https://docs.kernel.org/admin-guide/cgroup-v2.html`. Acesso em agosto de 2026.

Hwang, J., Zeng, S., y Wu, F., and Wood, T. (2013). A component-based performance comparison of four hypervisors. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 269–276. IEEE.

Kurtzer, G. M., Sochat, V., and Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):e0177459.

Linux Containers Org (2025). LXC – introduction. `https://linuxcontainers.org/lxc/introduction/`. Acesso em agosto de 2026.

Linux KVM Project (2014). KVM — kernel-based virtual machine. `https://linux-kvm.org/`. Acesso em agosto de 2026.

Matthews, J. N., Hu, W., Hapuarachchi, M., Deshane, T., Dimatos, D., Hamilton, G., McCabe, M., and Owens, J. (2007). Quantifying the performance isolation properties of virtualization systems. In *Proceedings of the 2007 Workshop on Experimental Computer Science (ExpCS '07)*, New York, NY, USA. ACM.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25. `https://www.cs.virginia.edu/stream/`.

Merrifield, T. and Taheri, H. R. (2016). Performance implications of extended page tables on virtualized x86 processors. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 25–35.

Morabito, R., Kjällman, J., and Komu, M. (2015). Hypervisors vs. lightweight virtualization: A performance comparison. In *2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 386–393. IEEE.

Open Container Initiative (2023). OCI runtime specification. `https://github.com/opencontainers/runtime-spec`. Acesso em agosto de 2026.

Pahl, C., Brogi, A., Soldani, J., and Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review. *IEEE Transactions on Cloud Computing*, 7(3):677–692.

Rao, J., Wang, K., Zhou, X., and Xu, C.-Z. (2013). Optimizing virtual machine scheduling in numa multicore systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 306–317. IEEE.

Red Hat, Inc. (2023). Podman documentation. `https://docs.podman.io/`. Acesso em agosto de 2026.

Red Hat, Inc. (2026). Building, running, and managing containers. `https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/10/pdf/building_running_and_managing_containers/Red_Hat_Enterprise_Linux-10-Building_running_and_managing_containers-en-US.pdf`.

Russell, R. (2008). virtio: Towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Operating Systems Review*, 42(5):95–103.

Scrivano, G. (2023). A fast and lightweight fully featured oci runtime and c library for running containers. `https://github.com/containers/crun`. Acesso em agosto de 2026.

Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. C. (2016). Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference (Middleware '16)*, Middleware '16, pages 1–13, New York, NY, USA. ACM.

Sharma, P. and Kulkarni, P. (2012). Singleton: system-wide page deduplication in virtual environments. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, pages 15–26.

Snell, Q. O., Mikler, A. R., and Gustafson, J. L. (1996). Netpipe: A network protocol independent performance evaluator. In *Proceedings of the IASTED International Conference on Intelligent Information Management and Systems*.

Torrez, A., Randles, T., and Priedhorsky, R. (2019). HPC container runtimes have minimal or no performance impact. In *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pages 37–42. IEEE.

Turnbull, J. (2014). *The Docker Book: Containerization Is the New Virtualization*. James Turnbull.

Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and De Rose, C. A. F. (2013). Performance evaluation of container-based virtualization for high performance computing environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 233–240. IEEE.

Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., and Zhou, W. (2018). A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 178–185. IEEE.