

Avaliação de Lustre e GekkoFS sob Cinco Estratégias de I/O em uma Aplicação CFD

João V. Vargas¹, Cristiano A. Künas¹, Philippe O. A. Navaux¹, Bruno M. Morales¹

¹Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{jvvoliveira, cakunas, navaux, bmmorales}@inf.ufrgs.br

Abstract. *High-Performance Computing (HPC) systems rely on parallel file systems capable of sustaining high throughput and low latency. Modern workloads, characterized by metadata-intensive operations and high concurrency, challenge traditional systems such as Lustre, whose centralized metadata server is a structural bottleneck. GekkoFS addresses this limitation by distributing metadata and data across compute nodes. This paper evaluates both systems in two scenarios. With the MDTest and IOR benchmarks, GekkoFS outperforms Lustre in metadata operations by up to one order of magnitude, while Lustre remains competitive in sequential throughput. With a real multi-GPU Computational Fluid Dynamics (CFD) solver, five write patterns are compared. On Lustre, asynchronous multi-file writes yield the lowest overhead (6.6%), whereas single-file consolidation yields the highest (53.7%), showing that the bottleneck moves from the metadata server to the data server. On GekkoFS, single-file asynchronous modes are the most efficient (12.0% and 12.3%).*

Resumo. *Sistemas de Computação de Alto Desempenho (HPC) dependem de sistemas de arquivos paralelos capazes de sustentar alto throughput e baixa latência. Cargas de trabalho modernas, caracterizadas por operações intensivas de metadados e alta concorrência, desafiam sistemas tradicionais como o Lustre, cujo servidor de metadados centralizado constitui um gargalo estrutural. O GekkoFS responde a essa limitação distribuindo metadados e dados nos nós de computação. Este artigo avalia ambos os sistemas em dois cenários. Com os benchmarks MDTest e IOR, o GekkoFS supera o Lustre em operações de metadados por até uma ordem de magnitude, enquanto o Lustre permanece competitivo em throughput sequencial. Com uma aplicação de Dinâmica de Fluidos Computacional (CFD) multi-GPU, cinco padrões de escrita são comparados. No Lustre, a escrita assíncrona com múltiplos arquivos apresenta o menor overhead (6,6%), ao passo que a consolidação em arquivo único apresenta o maior (53,7%), evidenciando o deslocamento do gargalo do servidor de metadados para o servidor de dados. No GekkoFS, os modos assíncronos com arquivo único são os mais eficientes (12,0% e 12,3%).*

O presente trabalho foi apoiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Código de Financiamento 001, pelo edital CNPq/MCTI/FNDCT Universal sob número 408755/2025-3 e pela Petrobras sob número 2020/00182-5. Os experimentos foram realizados na infraestrutura experimental Grid'5000, apoiada por um grupo de interesse científico hospedado pelo Inria e que inclui CNRS, RENATER e diversas universidades e organizações.

1. Introdução

Aplicações científicas de larga escala dependem de infraestruturas de Computação de Alto Desempenho (HPC) para processar e armazenar volumes de dados que frequentemente atingem a escala de petabytes: experimentos como o Large Hadron Collider geram aproximadamente 100 PB de dados e simulações multifísicas como FLASH e NyX operam na faixa de ≈ 10 PB cada [Ravi et al. 2023]. À medida que o poder computacional dos supercomputadores cresce, a capacidade de mover e acessar esses dados com eficiência torna-se fator determinante da produtividade científica.

Estudos empíricos conduzidos em supercomputadores de escala *petascale* indicam que o I/O é um dos principais gargalos em HPC. Aplicações típicas gastam entre 15% e 40% do tempo total de execução em operações de entrada e saída [Tarraf et al. 2024], e a análise de aproximadamente um milhão de *jobs* executados nos supercomputadores Intrepid, Mira e Edison ao longo de seis anos revelou que cerca de um terço deles gasta mais tempo em funções de metadados do que na transferência efetiva de dados [Luu et al. 2015]. A situação é agravada pela natureza dinâmica do desempenho de I/O: um ano de monitoramento contínuo em dois centros de classe *leadership* mostrou que atualizações de software, ocupação do sistema de arquivos e contenção entre jobs introduzem degradações que podem se estender por semanas [Lockwood et al. 2018], com diferenças de até 200 vezes no tempo de execução de cargas idênticas conforme o momento em que são executadas.

O Lustre [Cluster File Systems, Inc. 2007] é o sistema de arquivos com maior participação em HPC, presente em aproximadamente 20% dos sistemas do ranking Top500 [Carneiro et al. 2023]. Sua arquitetura baseia-se em um Management Server (MGS), um Metadata Server (MDS) centralizado, responsável por todas as operações de namespace, e em múltiplos Object Storage Servers (OSS) para o armazenamento distribuído dos dados. Essa separação oferece alto throughput para I/O sequencial de grandes blocos, mas introduz uma limitação estrutural: mais de 50% dos acessos a sistemas de arquivos distribuídos correspondem a operações de metadados [Laitala 2017] e um único MDS concentra toda essa carga. A análise do supercomputador Santos Dumont evidenciou ainda desequilíbrios de carga entre Object Storage Targets (OSTs) de até 300% da média e utilização de largura de banda que raramente supera 42% do pico teórico da instalação.

O GekkoFS [Vef et al. 2018, Vef et al. 2020] representa uma abordagem alternativa para cenários efêmeros. Classificado como sistema de arquivos ad-hoc, distribui metadados e dados diretamente nos nós de computação durante a execução do job, sem servidor de metadados centralizado, e intercepta chamadas POSIX via `LD_PRELOAD`, o que dispensa modificações de código na aplicação. Em experimentos com até 512 nós atingiu aproximadamente 46 milhões de operações de *create* por segundo, superando o Lustre por ordens de magnitude. Em contrapartida, não oferece persistência além da vida do job e apresenta restrições de compatibilidade com a semântica POSIX completa.

Além da escolha do sistema de arquivos, o padrão de escrita adotado pela aplicação constitui um segundo eixo determinante de desempenho, frequentemente negligenciado nas avaliações comparativas. No modelo síncrono a aplicação aguarda a conclusão de cada escrita antes de prosseguir, garantindo durabilidade mas desperdiçando ciclos de processamento. No modelo assíncrono as operações são submetidas em segundo

plano, permitindo sobrepor computação e I/O e potencialmente ocultar a latência do armazenamento [Ravi et al. 2023, Tarraf et al. 2024]. O assincronismo, contudo, tem custo próprio: a submissão irrestrita de requisições pode sobrecarregar o sistema de arquivos compartilhado e degradar o desempenho de outros jobs da plataforma [Tarraf et al. 2024].

Este artigo avalia comparativamente Lustre e GekkoFS combinando essas duas dimensões. Primeiro, os benchmarks sintéticos MDTest e IOR [IOR Project sd] caracterizam o comportamento arquitetural de cada sistema em metadados, throughput e IOPS. Em seguida, a análise é estendida a uma aplicação real de Dinâmica de Fluidos Computacional (CFD) acelerada por GPU, na qual se avaliam cinco padrões de escrita: síncrono com múltiplos arquivos (*naive*), assíncrono com múltiplos arquivos (*async*), síncrono com arquivo único por checkpoint (*savez*) e dois modos assíncronos com arquivo único, um com fila limitada (*hybrid*) e outro com fila livre (*async_npz*), sendo os dois últimos propostos nesse trabalho.

As principais contribuições são: (i) a caracterização empírica do deslocamento do gargalo do MDS para o OSS no Lustre, com evidência de que a consolidação de arquivos pode piorar o desempenho mesmo reduzindo em 4 vezes o número de operações de metadados, (ii) a demonstração de que a escolha do padrão de I/O responde por uma diferença de até 44% do tempo de computação pura em uma aplicação real, magnitude comparável à da própria troca de sistema de arquivos e (iii) a proposta e avaliação do modo *hybrid*, escrita assíncrona com fila de profundidade limitada, como alternativa estável ao assincronismo irrestrito.

O restante do texto está organizado como segue. A Seção 2 apresenta a fundamentação, a Seção 3 discute os trabalhos relacionados, a Seção 4 descreve a metodologia, a Seção 5 apresenta e discute os resultados e a Seção 6 conclui o trabalho.

2. Fundamentação

2.1. Lustre e GekkoFS

O Lustre é um sistema de arquivos paralelo de propósito geral que separa explicitamente o plano de metadados do plano de dados. O Metadata Server (MDS), apoiado por um ou mais Metadata Targets (MDTs), atende às operações de namespace, enquanto os Object Storage Servers (OSS), apoiados pelos Object Storage Targets (OSTs), armazenam o conteúdo dos arquivos. O cliente é implementado como módulo de *kernel* e apresenta uma visão POSIX completa e coerente. O conteúdo de cada arquivo pode ser distribuído entre múltiplos OSTs por *striping*, o que permite agregar a largura de banda de vários servidores em uma única operação e favorece cargas dominadas por transferências sequenciais de grandes blocos. O caminho de metadados, porém, permanece serializado no MDS, o que se torna limitante quando muitos clientes criam ou consultam arquivos simultaneamente. Estudos de produção confirmam ainda que o desempenho degrada quando os OSTs ultrapassam 90% de ocupação [Lockwood et al. 2018].

O GekkoFS é um sistema de arquivos ad-hoc, instanciado sob demanda para a duração de um job e desmontado ao seu término. Cada nó participante executa um *daemon* que armazena dados em dispositivos locais e metadados em uma instância local do RocksDB. Não existe servidor de metadados dedicado: o namespace é particionado entre todos os daemons por uma tabela de *hash* distribuída, de modo que a capacidade de processamento de metadados cresce linearmente com o número de nós. Do lado do cliente,

uma biblioteca carregada via `LD_PRELOAD` intercepta as chamadas de sistema relacionadas a arquivos e as redireciona ao daemon responsável. Em troca desse desempenho, o GekkoFS relaxa parte da semântica POSIX e não oferece persistência além do tempo de vida do job, o que o posiciona como camada de *burst buffer* e não como substituto de armazenamento permanente.

2.2. Padrões de escrita em HPC

Aplicações HPC de larga escala alternam periodicamente fases de computação e de persistência de resultados ou checkpoints [Ravi et al. 2023]. No modelo síncrono, a aplicação submete a escrita e aguarda sua conclusão, de modo que o tempo de cada iteração é $t_{iter} = t_{comp} + t_{io}$. No modelo assíncrono, a escrita é delegada a uma *thread* ou a um *pool* de *threads* e a computação do passo seguinte inicia imediatamente, resultando em $t_{iter} = \max(t_{comp}, t_{io}) + t_{ovh}$, onde t_{ovh} representa o custo de submissão e sincronização. Quando o tempo de I/O excede o de computação, a sobreposição é apenas parcial e resta espera ao final da execução.

Uma dimensão ortogonal é a granularidade dos arquivos gerados por checkpoint. No padrão de múltiplos arquivos, cada campo da simulação é escrito em um arquivo separado por *rank*, a consolidação em arquivo único reduz proporcionalmente o número de operações de metadados, e a hipótese natural é que essa redução beneficie sistemas com MDS centralizado. Como se mostra na Seção 5, essa hipótese não se sustenta quando o plano de dados é o recurso escasso.

3. Trabalhos Relacionados

Caracterização de I/O em produção. Luu et al. [Luu et al. 2015] conduziram o primeiro estudo multiplataforma de comportamento de I/O em escala, analisando *logs* do Darshan de aproximadamente um milhão de jobs ao longo de seis anos. Os resultados revelam subutilização sistemática, com três quartos das aplicações jamais atingindo 1 GB/s de *throughput* agregado. Lockwood et al. [Lockwood et al. 2018] monitoraram dois centros de classe *leadership* durante um ano com o *framework* TOKIO e demonstraram que o desempenho de I/O não é estacionário, com variações de longo prazo correlacionadas à ocupação dos OSTs coexistindo com degradações transientes por contenção entre jobs. Carneiro et al. [Carneiro et al. 2023] caracterizaram o Lustre do Santos Dumont durante três meses, evidenciando desequilíbrio de carga entre OSTs superior a 300% da média em picos. Esses três trabalhos documentam o problema no nível da plataforma. O presente artigo avalia soluções arquiteturais concretas sob condições controladas.

Comparação de sistemas de arquivos distribuídos. Depardon et al. [Depardon et al. 2013] analisam seis sistemas distribuídos no Grid'5000, a mesma plataforma utilizada aqui, destacando as diferenças entre arquiteturas de metadados centralizadas e distribuídas, mas sem contemplar sistemas *ad-hoc*, inexistentes à época. Laitala [Laitala 2017] confirma que arquiteturas centralizadas enfrentam contenção crescente com o número de clientes e de arquivos, enquanto arquiteturas distribuídas escalam melhor ao custo de maior complexidade de consistência. Vef et al. [Vef et al. 2018, Vef et al. 2020] apresentam o GekkoFS e o avaliam em até 512 nós do MOGON II, com cerca de 46 milhões de operações de *create* e 44 milhões de *stat* por segundo, além de ganhos de 7% a 9% no tempo de execução do *solver* OpenFOAM em relação ao Lustre. Esse trabalho é o ponto de partida direto desta avaliação, com a distinção de que o OpenFOAM realiza I/O de forma síncrona e implícita, sem permitir comparação entre padrões

de escrita. Astsatryan et al. [Astsatryan et al. 2026] atacam o gargalo de metadados do Lustre em cargas de aprendizado profundo pela convergência com armazenamento de objetos compatível com S3, mantendo o Lustre como substrato e adicionando uma camada de *gateway*, a alternativa aqui avaliada é estruturalmente distinta, pois o paradigma ad-hoc não requer infraestrutura além dos nós já alocados ao job.

I/O assíncrono. Ravi et al. [Ravi et al. 2023] avaliam I/O assíncrono em aplicações de diferentes domínios usando as implementações de alto nível do HDF5 [Folk et al. 2011] e do ADIOS2 [Godoy et al. 2020], demonstrando que a sobreposição pode ocultar parcial ou totalmente o custo de escrita quando a computação supera o I/O. Tarraf et al. [Tarraf et al. 2024] tratam de um aspecto frequentemente negligenciado, a determinação da largura de banda necessária para que as escritas em segundo plano sejam concluídas dentro da janela de computação seguinte, e demonstram que aplicações assíncronas frequentemente consomem mais largura de banda do que o necessário, prejudicando jobs concorrentes. Esse resultado motiva diretamente o modo *hybrid* proposto nesse trabalho.

Posicionamento. Os trabalhos revisados evidenciam três lacunas: estudos de caracterização documentam os problemas do Lustre em produção sem avaliar alternativas arquiteturais, avaliações do GekkoFS tratam o sistema de arquivos de forma isolada, sem explorar o padrão de escrita como dimensão independente, e estudos de I/O assíncrono operam exclusivamente sobre sistemas convencionais, sem avaliar a interação entre padrão de escrita e sistema ad-hoc. Este trabalho combina as três dimensões em um único estudo experimental.

4. Metodologia

Os experimentos foram conduzidos na plataforma Grid’5000 [Balouek et al. 2013, Grid’5000 2026], infraestrutura experimental de larga escala operada por um consórcio de instituições francesas. A característica que motivou essa escolha é o modelo de reserva exclusiva, no qual pesquisadores reservam nós com controle total sobre hardware e software. Esse controle é crítico em experimentos de I/O, cuja variabilidade em plataformas de produção pode atingir ordens de magnitude [Lockwood et al. 2018]. A Tabela 1 resume os dois aglomerados utilizados.

Tabela 1. Especificações dos aglomerados utilizados no Grid’5000.

Componente	<i>paradoxe</i> (Rennes)	<i>chiffot</i> (Lille)
Processador	2 × Xeon Gold 5320	2 × Xeon Gold 6126
Núcleos por nó	52 físicos	24 físicos
Memória RAM	384 GiB	192 GiB DDR4
Armazenamento	2 × SSD de 1,92 TB	1 SSD de 480 GB, 4 HDDs de 4 TB
GPUs	sem GPU	2 × NVIDIA Tesla P100 16 GiB
Rede	25 Gbps Ethernet	2 × 25 Gbps Ethernet

4.1. Primeiro experimento: benchmarks sintéticos

O primeiro experimento foi conduzido no aglomerado *paradoxe*, com 6 nós alocados e distribuídos de forma distinta para cada sistema. No Lustre, 1 nó atuou como MDS, 2

nós como OSS e 3 nós como clientes. No GekkoFS, 3 nós atuaram como *daemons*, armazenando metadados e dados localmente, e 3 nós como clientes. Essa topologia reflete as recomendações de implantação de cada sistema: o Lustre separa os papéis de metadados e dados em nós dedicados, enquanto o GekkoFS distribui ambos igualmente entre os *daemons*.

Para a avaliação de metadados foi utilizado o MDTest, que mede as taxas de criação, consulta de atributos e remoção de arquivos sob concorrência, executado com 4 processos MPI por nó cliente e variando o número de clientes de 1 a 3. Para throughput e IOPS foi utilizado o IOR em operações sequenciais de leitura e escrita, com blocos de 64 KB, 1 MB e 4 MB, faixa que cobre desde operações pequenas, que tendem a saturar o número de IOPS, até transferências maiores, que tendem a saturar a largura de banda da rede.

4.2. Segundo experimento: aplicação CFD

O segundo experimento foi conduzido no aglomerado *chiffnot*, cuja presença de duas GPUs por nó o torna adequado à simulação acelerada utilizada como carga de trabalho. Foram alocados 6 nós para cada configuração. No GekkoFS, 2 nós atuaram como *daemons*, armazenando dados nos HDDs locais via POSIX e metadados via RocksDB, e 4 nós como clientes de computação com o LD_PRELOAD ativado. No Lustre, 1 nó atuou como MGS e MDS combinados, com o MDT sobre o SSD local, 1 nó como OSS com um único OST sobre um dos HDDs, e 4 nós como clientes de computação montando o sistema via módulo de kernel. A configuração de um único OSS reflete um cenário conservador, compatível com os recursos disponíveis e suficiente para observar os efeitos de serialização no plano de dados, que se mostram centrais na análise.

4.2.1. Carga de trabalho

A carga de trabalho é o *solver* AI4PDEs [Wang et al. 2024], implementação multi-GPU e multinó das equações de Navier-Stokes incompressíveis desenvolvida em PyTorch com comunicação inter-GPU via NVIDIA Collective Communications Library *NCCL*, que resolve

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} = -\nabla p + Re^{-1} \nabla^2 \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0, \quad (1)$$

onde $\mathbf{u}$ é o campo de velocidade, p o campo de pressão e Re o número de Reynolds. A implementação utiliza a infraestrutura de redes neurais convolucionais do PyTorch para representar operadores de diferenças finitas com pesos fixos: gradientes e Laplaciano são convoluções tridimensionais com núcleos $3 \times 3 \times 3$. A equação de Poisson para a pressão é resolvida por um ciclo *multigrid* do tipo F e a integração temporal segue um esquema predictor-corretor de dois passos. Em múltiplos nós, o domínio é decomposto entre os *ranks*, com troca de halo antes de cada convolução.

A cada n_{out} passos de tempo, cada *rank* escreve os campos u , v , w e p do seu subdomínio local, sem operação de agrupamento e sem comunicação adicional na fase de I/O. Esse padrão, muitos arquivos escritos concorrentemente por múltiplos processos, é exatamente o cenário que pressiona o MDS centralizado do Lustre e para o qual o GekkoFS foi projetado. A Tabela 2 resume a configuração adotada: a decomposição 1D-Z com grade de 512^3 células resulta em subdomínios locais de $512 \times 512 \times 64$ células

por *rank* e cada campo ocupa 64 MiB em precisão simples, de modo que cada *checkpoint* escreve 256 MiB por *rank*, totalizando 4 GiB por execução completa.

Tabela 2. Configuração experimental da aplicação AI4PDEs.

Parâmetro	Valor	Parâmetro	Valor
Grid	512 ³ células	Passos de tempo	20
Decomposição	1D-Z, 8 ranks	Frequência de I/O	a cada 10 passos
Computação	4 nós × 2 GPUs	Checkpoints/execução	2
Troca de halo	bloqueante	Repetições/modo	10

4.2.2. Modos de I/O avaliados

Foram avaliados cinco modos de I/O, selecionáveis por argumento de linha de comando, que diferem no padrão de serialização dos dados, arquivos separados ou arquivo único, e na estratégia de submissão, síncrona ou assíncrona, formando uma grade experimental que permite isolar o efeito de cada dimensão. A Tabela 3 resume suas características.

Tabela 3. Características dos modos de I/O avaliados.

Modo	Serialização	Submissão	Arquivos/execução	Fila
<i>none</i>	sem I/O	sem I/O	0	sem fila
<i>naive</i>	4 × .npy	síncrona	64	sem fila
<i>async</i>	4 × .npy	assíncrona	64	livre
<i>savez</i>	1 × .npz	síncrona	16	sem fila
<i>hybrid</i>	1 × .npz	assíncrona	16	limitada (1)
<i>async_npz</i>	1 × .npz	assíncrona	16	livre

O modo *none* não realiza escrita. Ele isola o tempo de computação pura, estabelecendo a linha de base de comparação do overhead, a partir da qual os demais modos são calculados. O modo *naive* implementa escrita síncrona, com cada *rank* chamando `np.save()` quatro vezes por checkpoint e aguardando o retorno de cada chamada. O modo *async* submete as mesmas quatro chamadas a um `ThreadPoolExecutor` com um *worker* por *rank*, retornando imediatamente. Ao final da simulação o executor é drenado e o tempo de espera é registrado separadamente. O modo *savez* consolida os quatro campos em um único arquivo .npz por *rank* e por checkpoint, de forma síncrona, reduzindo o número de criações de arquivo de 64 para 16 por execução.

Os modos *hybrid* e *async_npz*, propostos neste trabalho, combinam a consolidação em arquivo único com submissão assíncrona e diferem apenas na política de controle da fila de escritas pendentes. No *async_npz* a fila é ilimitada: se o sistema de arquivos não conseguir drenar as escritas entre checkpoints, as operações se acumulam e o tempo de espera final cresce proporcionalmente. No *hybrid* há um controle de fila de profundidade unitária, no qual o *rank* verifica se o checkpoint anterior já foi concluído antes de submeter o próximo e, em caso negativo, aguarda sua conclusão. Esse mecanismo impede o acúmulo irrestrito de operações pendentes sem bloquear completamente a computação

quando o I/O é mais rápido que o passo de tempo, combinando a sobreposição do modelo assíncrono com a previsibilidade do modelo síncrono.

4.2.3. Protocolo e hipóteses

Para cada combinação de sistema de arquivos e modo de I/O foram executadas 10 repetições independentes, em job com alocação exclusiva dos 6 nós. A independência entre execuções é garantida pela reinicialização do sistema de arquivos: no GekkoFS os *daemons* são reiniciados, eliminando dados residuais, no Lustre o diretório de saída é recriado vazio. As métricas coletadas são o tempo total de execução, o tempo acumulado nas chamadas de escrita, que no caso assíncrono corresponde apenas ao tempo de submissão, e o tempo de drenagem da fila ao final da simulação.

Duas hipóteses orientaram o desenho experimental. A hipótese de metadados prevê que a consolidação em arquivo único reduzirá o *overhead* no Lustre ao diminuir a carga sobre o MDS centralizado, com impacto neutro no GekkoFS, que não possui MDS. A hipótese de assincronismo prevê que a submissão assíncrona reduzirá o overhead em ambos os sistemas ao sobrepor computação e I/O, com maior benefício no sistema de maior latência. Os modos *hybrid* e *async_npz* nascem da combinação dessas hipóteses, isolando o efeito do assincronismo no contexto da consolidação de arquivos.

5. Resultados

5.1. Benchmarks sintéticos

A Figura 1 apresenta os resultados do MDTest para criação, consulta e remoção de arquivos, em escala logarítmica, variando de 1 a 3 nós com 4 processos por nó. O GekkoFS supera o Lustre nas três operações e a vantagem se acentua com o aumento do número de nós. Com 1 nó ambos operam em faixas próximas, na ordem de 10^4 operações por segundo, ao escalar para 3 nós o GekkoFS aproxima-se de 10^5 operações por segundo em todas as operações, enquanto o Lustre permanece próximo de 10^4 . O ponto relevante não é apenas a diferença absoluta, mas a inclinação das curvas: o Lustre praticamente não escala com o número de clientes, pois o MDS único satura, ao passo que o GekkoFS cresce de forma aproximadamente linear com o número de *daemons*. Isso é particularmente relevante para cargas de aprendizado de máquina, nas quais checkpoints frequentes e criação de arquivos temporários geram exatamente o padrão que penaliza arquiteturas com metadados centralizados.

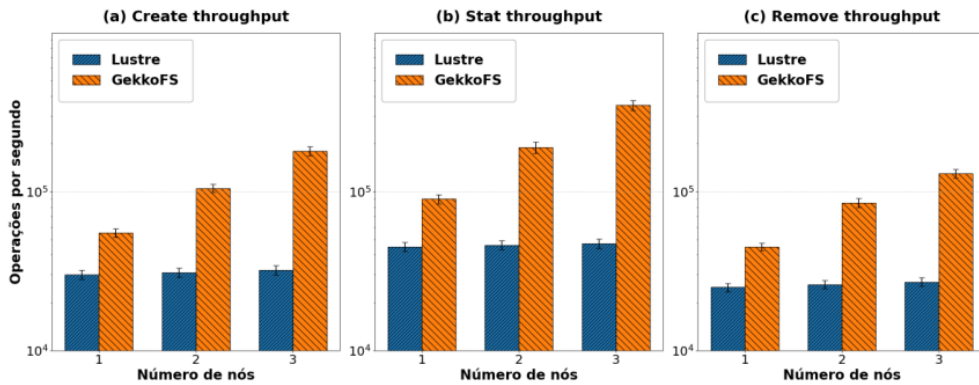


Figura 1. Desempenho do MDTest com quatro processos por nó, variando o número de nós.

A Figura 2 revela um cenário distinto para o throughput de dados. Em escrita sequencial o Lustre apresenta desempenho competitivo ou superior, sobretudo com 1 nó, com 3 nós a diferença se reduz, mas o Lustre mantém ligeira vantagem, resultado esperado dado que o GekkoFS armazena dados localmente sem as otimizações de agregação de escrita presentes no cliente Lustre. Em leitura sequencial o Lustre apresenta throughput superior na maioria dos tamanhos de bloco, reflexo do *striping* entre OSTs, que explora paralelismo durante o acesso e favorece cargas dominadas por leitura de grandes volumes. Vale notar que a escala logarítmica suaviza diferenças absolutas: ambos operam na faixa de 10^3 MB/s, o que indica que para cargas sequenciais em larga escala a diferença prática entre os dois sistemas é menos crítica do que no caso dos metadados.

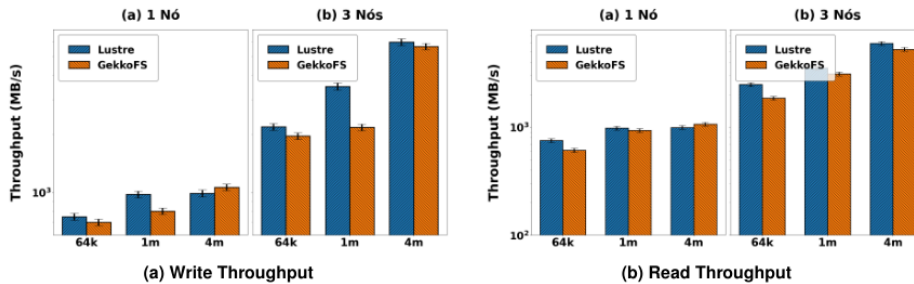


Figura 2. Throughput de Lustre e GekkoFS em escrita e leitura sequenciais.

A Figura 3 apresenta o número de operações de I/O por segundo. Em escrita, o Lustre apresenta IOPS superior para blocos pequenos com 1 nó, em razão do uso de cache e agregação pelo cliente, com 3 nós o desempenho do GekkoFS se aproxima ou supera o do Lustre, indicando que o paralelismo distribuído compensa a ausência dessas otimizações. Em leitura, o Lustre obtém bom desempenho para blocos pequenos e, à medida que o tamanho do bloco cresce, os resultados convergem, com o GekkoFS superior nos blocos maiores. A leitura conjunta das três figuras sugere uma divisão de responsabilidades bem definida: o GekkoFS domina o plano de metadados, o Lustre mantém vantagem no plano de dados sequenciais e ambos convergem em cargas de blocos grandes com alta concorrência. Essa divisão é o ponto de partida do segundo experimento, no qual se investiga qual dos dois planos determina o desempenho de uma aplicação real.

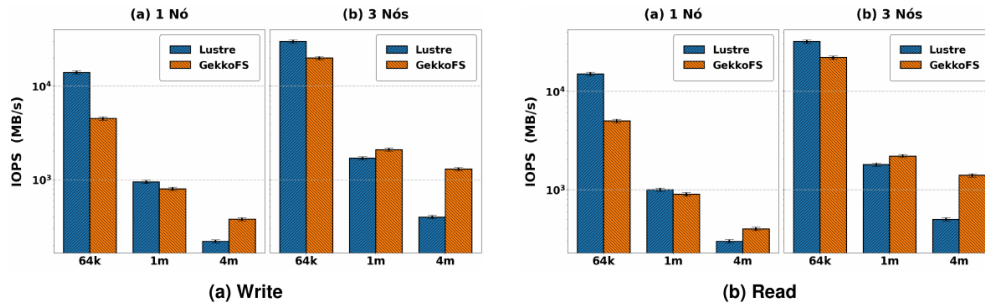


Figura 3. Comparação de IOPS entre Lustre e GekkoFS para 1 e 3 nós.

5.2. Aplicação CFD

A Figura 4 apresenta o tempo total de execução da simulação para cada modo de I/O nos dois sistemas de arquivos. A linha de base sem I/O revela custo computacional praticamente equivalente entre eles, com $10,04 \text{ s} \pm 0,04 \text{ s}$ no GekkoFS e $9,83 \text{ s} \pm 0,03 \text{ s}$ no Lustre. Essa equivalência é importante porque garante que as diferenças observadas nos demais modos são atribuíveis exclusivamente ao subsistema de I/O.

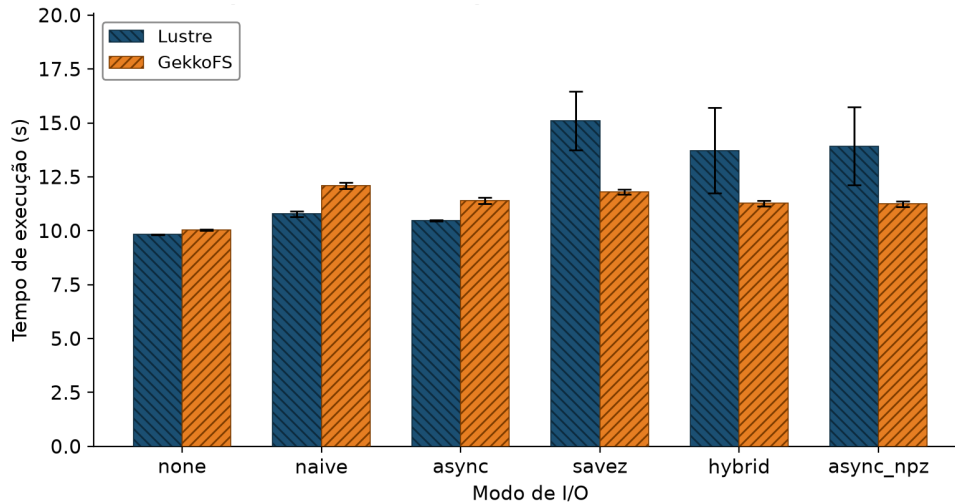


Figura 4. Tempo total de execução da simulação CFD por modo de I/O.

A partir do momento em que operações de escrita são introduzidas, os dois sistemas divergem de forma expressiva. No Lustre, os modos *naive* e *async* apresentam overhead moderado, de 9,7% e 6,6% respectivamente. Os modos *savez*, *hybrid* e *async_npz*, por outro lado, elevam o tempo de execução para 15,09 s (mais 53,7%), 13,72 s (mais 39,7%) e 13,93 s (mais 41,8%), com variância substancialmente maior. Esse comportamento bimodal indica que o Lustre não responde de forma determinística à escrita de arquivos grandes e únicos por *rank*: em algumas execuções a escrita é relativamente rápida, em outras o OSS fica congestionado e bloqueia a simulação por vários segundos. O GekkoFS, em contraste, mantém comportamento estável em todos os modos, com overhead entre 12,0% no *async_npz* e 20,4% no *naive* e desvio padrão sempre inferior a 0,16 s. A estabilidade decorre da escrita local nos dispositivos dos nós daemon, sem contenção de rede com um servidor de dados compartilhado.

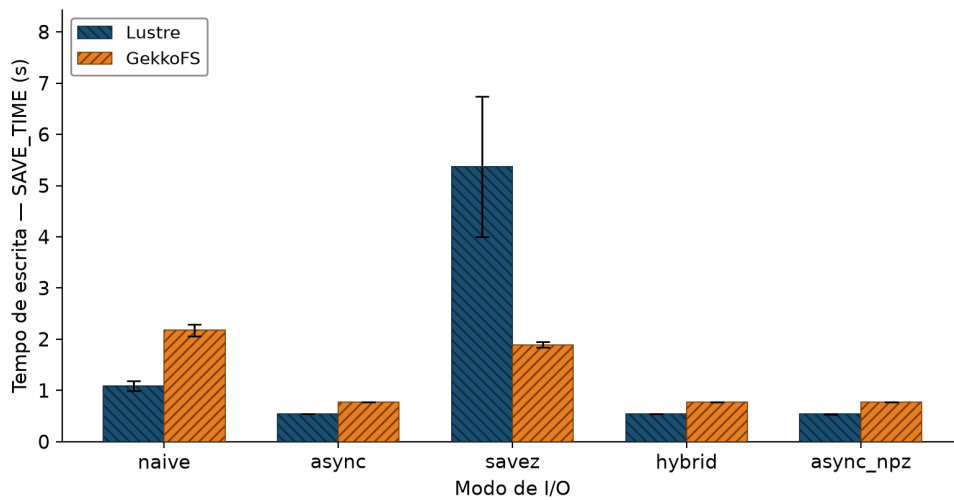


Figura 5. Tempo acumulado em operações de escrita por modo de I/O.

A Figura 5 isola o tempo efetivamente gasto nas operações de escrita, desconsiderando a computação, o que permite atribuir a cada modo a responsabilidade pelo overhead observado no tempo total. O comportamento mais notável é o do modo *savez* no Lustre, com tempo médio de escrita de $5,37 \text{ s} \pm 1,30 \text{ s}$, quase cinco vezes superior ao do modo *async* ($0,57 \text{ s}$) e mais de três vezes superior ao do modo *naive* ($1,09 \text{ s}$). No GekkoFS, o mesmo modo registra $1,93 \text{ s} \pm 0,05 \text{ s}$, sem variância expressiva.

5.3. Discussão

A hipótese de metadados é refutada. Embora a consolidação em arquivo único reduza o número de criações de arquivo de 64 para 16 por execução, uma redução de 4 vezes, o modo *savez* produziu o pior desempenho observado em todo o estudo. O gargalo não estava no MDS, mas na escrita serializada de um arquivo maior sobre o único OSS disponível: quatro arquivos menores escritos em paralelo por cada *rank* distribuem melhor a carga sobre o enlace de rede e sobre o *buffer* do OSS do que um único arquivo de tamanho equivalente. Otimizar o plano de metadados sem observar o plano de dados pode, portanto, piorar o desempenho, mesmo quando a métrica otimizada melhora conforme o previsto. No GekkoFS os tempos de escrita são consistentes entre todos os modos, reflexo da escrita local nos *daemons*, cujo throughput, entre aproximadamente 220 e 240 MB/s por nó, absorve qualquer um dos padrões avaliados.

A hipótese de assincronismo é confirmada apenas parcialmente. O assincronismo reduziu o overhead de forma clara nos modos com múltiplos arquivos, no Lustre de 9,7% para 6,6%. Nos modos de arquivo único, contudo, não houve benefício no Lustre: o arquivo consolidado é grande o suficiente para que a *thread* de segundo plano não conclua a escrita antes do próximo checkpoint, acumulando trabalho pendente e resultando em tempo de drenagem de $3,7 \text{ s} \pm 1,8 \text{ s}$ ao final da simulação. Esse resultado ilustra o fenômeno descrito por Tarraf et al. [Tarraf et al. 2024]: o assincronismo só oculta latência enquanto a largura de banda for suficiente para drenar a fila dentro da janela de computação, quando não é, o custo apenas se desloca para o final da execução.

O padrão de I/O ideal depende da arquitetura de armazenamento. A melhor estratégia para o Lustre foi o modo *async* com múltiplos arquivos, com overhead de ape-

nas 6,6%, pois a submissão assíncrona oculta a latência de escrita na janela de computação entre checkpoints e a distribuição em múltiplos arquivos aproveita a capacidade do OSS de receber fluxos paralelos de diferentes *ranks*. Para o GekkoFS, os melhores modos foram *async_npz* e *hybrid*, com overhead de 12,0% e 12,3%, uma vez que a escrita local torna o sistema praticamente indiferente ao número de arquivos por checkpoint. O modo *hybrid* atingiu desempenho equivalente ao *async_npz*, com a vantagem adicional de limitar por construção o acúmulo de escritas pendentes, propriedade desejável em plataformas compartilhadas nas quais a submissão irrestrita prejudica jobs concorrentes.

A magnitude do efeito justifica a atenção ao padrão de escrita. No Lustre, a diferença entre o melhor modo (*async*, 10,46 s) e o pior (*savez*, 15,09 s) é de 4,6 s, equivalente a 44% do tempo de computação pura, da mesma ordem de grandeza do ganho obtido pela substituição do sistema de arquivos. A vantagem prática é que alterar o padrão de escrita exige apenas modificações localizadas no código da aplicação, enquanto implantar um sistema de arquivos ad-hoc exige suporte da infraestrutura. A recomendação que decorre desses resultados é tratar a otimização do padrão de I/O como primeira medida, antes ou em conjunto com a avaliação de sistemas de arquivos alternativos.

Limitações. A configuração de um único OSS no Lustre limita o paralelismo de escrita e amplifica o efeito de serialização nos modos de arquivo único, em instalações com múltiplos OSSs e *striping* os valores quantitativos podem diferir, ainda que a conclusão qualitativa permaneça válida. A escala reduzida, com 4 nós de computação e 8 *ranks*, pode não refletir regimes de maior concorrência. O GekkoFS não oferece persistência além da vida do job, de modo que em fluxos reais seria necessário transferir os checkpoints para armazenamento permanente, custo não avaliado aqui. Por fim, o ambiente controlado do Grid'5000 pode superestimar o desempenho relativo do GekkoFS se comparado a ambientes de produção com contenção de rede [Lockwood et al. 2018].

6. Conclusão

Este trabalho avaliou comparativamente Lustre e GekkoFS em dois cenários complementares. Nos benchmarks sintéticos, o GekkoFS superou o Lustre em metadados por até uma ordem de magnitude ao escalar de 1 para 3 nós, enquanto o Lustre manteve throughput sequencial competitivo ou superior. Na aplicação CFD real, o modo assíncrono com múltiplos arquivos foi a melhor estratégia para o Lustre, com overhead de 6,6% sobre a linha de base sem I/O, ao passo que os modos assíncronos com arquivo único foram os mais eficientes no GekkoFS (12,0% e 12,3%), com variância uniformemente baixa.

É importante salientar a refutação da hipótese de que consolidar arquivos beneficiaria o Lustre ao aliviar o MDS: a consolidação reduziu em 4 vezes o número de criações de arquivo, mas produziu o pior desempenho observado, com 53,7% de overhead, porque o gargalo se deslocou do plano de metadados para o plano de dados. Otimizações guiadas por uma única métrica arquitetural podem, portanto, ser contraproducentes, o que reforça a necessidade de avaliar padrão de escrita e arquitetura de armazenamento em conjunto.

Como trabalhos futuros, pretende-se avaliar o Lustre com múltiplos OSSs e *striping*, replicar os experimentos em escala maior e estender a análise a outros sistemas efêmeros, como BeeGFS [ThinkParQ and Fraunhofer ITWM 2024] on Demand e DAOS [Liang et al. 2020].

Referências

- Astsatryan, H., Dovrolis, C., and Baghdasaryan, H. (2026). Convergence of HPC and cloud storage systems for deep learning workflows: An evaluation of LustreFS and S3-compatible object storage. *IEEE Access*.
- Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., and Sarzyniec, L. (2013). Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer.
- Carneiro, A. R., Bez, J. L., Boito, F. Z., Fagundes, B. A., Osthoff, C., and Navaux, P. O. A. (2023). Uncovering I/O demands on HPC platforms: Peeking under the hood of Santos Dumont. *Journal of Parallel and Distributed Computing*, 182:104744.
- Cluster File Systems, Inc. (2007). Lustre: A scalable, high-performance file system. White paper, Cluster File Systems, Inc.
- Depardon, B., Le Mahec, G., and Séguin, C. (2013). Analysis of six distributed file systems. Technical Report hal-00789086, HAL.
- Folk, M., Heber, G., Koziol, Q., Pourmal, E., and Robinson, D. (2011). An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA. Association for Computing Machinery.
- Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P., Choi, J., Germaschewski, K., Huck, K., Huebl, A., Kim, M., Kress, J., Kurc, T., Liu, Q., Logan, J., Mehta, K., Ostrouchov, G., Parashar, M., Poeschel, F., Pugmire, D., Suchyta, E., Takahashi, K., Thompson, N., Tsutsumi, S., Wan, L., Wolf, M., Wu, K., and Klasky, S. (2020). ADIOS 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561.
- Grid'5000 (2026). Grid'5000: Home. <https://www.grid5000.fr>. Acesso em: 19 jun. 2026.
- IOR Project (s.d.). IOR and mdtest: Parallel I/O and metadata benchmarks. <https://github.com/hpc/ior>. Repositório oficial. Acesso em: 19 out. 2026.
- Laitala, J. (2017). Metadata management in distributed file systems. Master's thesis, University of Helsinki.
- Liang, Z., Lombardi, J., Chaarawi, M., and Hennecke, M. (2020). DAOS: A scale-out high performance storage stack for storage class memory. In *Supercomputing Frontiers*, volume 12082 of *Lecture Notes in Computer Science*, pages 40–54, Cham. Springer.
- Lockwood, G. K., Snyder, S., Wang, T., Byna, S., Carns, P., and Wright, N. J. (2018). A year in the life of a parallel file system. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 931–943. IEEE.
- Luu, H., Winslett, M., Gropp, W., Ross, R., Carns, P., Harms, K., Prabhat, M., Byna, S., and Yao, Y. (2015). A multiplatform study of I/O behavior on petascale supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 33–44.

- Ravi, J., Byna, S., Koziol, Q., Tang, H., and Becchi, M. (2023). Evaluating asynchronous parallel I/O on HPC systems. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 211–221. IEEE.
- Tarraf, A., Bandet, A., Boito, F., Pallez, G., and Wolf, F. (2024). I/O behind the scenes: Bandwidth requirements of HPC applications with asynchronous I/O. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 426–439. IEEE.
- ThinkParQ and Fraunhofer ITWM (2024). *BeeOND: BeeGFS On Demand*. ThinkParQ. Documentação do BeeGFS, versão 7.2.13. Acesso em: 19 out. 2026.
- Vef, M.-A., Moti, N., Süß, T., Tacke, M., Tocci, T., Nou, R., Miranda, A., Cortes, T., and Brinkmann, A. (2020). GekkoFS: A temporary burst buffer file system for HPC applications. *Journal of Computer Science and Technology*, 35:72–91.
- Vef, M.-A., Moti, N., Süß, T., Tocci, T., Nou, R., Miranda, A., Cortes, T., and Brinkmann, A. (2018). GekkoFS: A temporary distributed file system for HPC applications. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 319–324. IEEE.
- Wang, Y., Chen, B., Chen, C., Zhu, J., Xiang, B., Heaney, C. E., and Pain, C. C. (2024). Artificial intelligence for partial differential equations in computational mechanics: A review. *Applied Mechanics Reviews*, pages 1–81.