Escalando a E/S do SDDP em Ambientes HPC: da Escrita Serializada às Escritas Paralelas com MPI-IO sobre Lustre*

Marcelo Barros da Cruz[1], **Claudio Luis de Amorim**[1], **Diego Leonel Cadette Dutra**[1]

[1]Programa de Engenharia de Sistemas e Computação (PESC) — COPPE
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro — RJ — Brasil

{mbrcruz, amorim, ddutra}@cos.ufrj.br

Abstract. *Efficient I/O scalability remains a major challenge for parallel applications on HPC environments, and is particularly critical in energy-sector optimization models such as Stochastic Dual Dynamic Programming (SDDP), which require frequent, distributed access to large datasets. This paper investigates the I/O bottlenecks of the final simulation phase of SDDP and proposes an MPI-IO architecture over Lustre that replaces the current single-writer strategy. Evaluated on AWS (FSx for Lustre) and the Santos Dumont supercomputer (LNCC), from 2 to 32 nodes, at 32 nodes it reduces total runtime by up to 36.7% (AWS) and 40.4% (SDumont), with perceived aggregate bandwidth gains of 113× and 13.7×, respectively.*

Resumo. *A escalabilidade eficiente de E/S é um desafio crítico em aplicações paralelas de alto desempenho, sobretudo em modelos de otimização do setor energético como o SDDP, que demandam acesso frequente e distribuído a grandes conjuntos de dados. Este artigo investiga os gargalos de E/S da fase de simulação final do SDDP e propõe uma arquitetura baseada em MPI-IO sobre Lustre, substituindo a estratégia atual com escritor único. Avaliada nos ambientes AWS (FSx for Lustre) e Santos Dumont (LNCC), de 2 a 32 nós, reduz o tempo total em até 36,7% (AWS) e 40,4% (SDumont) em 32 nós, com ganhos de banda agregada percebida de 113× e 13,7×, respectivamente.*

1. Introdução

A computação de alto desempenho (*High-Performance Computing* — HPC) tornou-se ferramenta indispensável para aplicações científicas que processam volumes crescentes de dados [Kang et al. 2020]. Nesses ambientes, o desempenho global de uma aplicação não depende apenas da capacidade computacional dos processadores, mas também da eficiência das operações de entrada e saída (E/S), que frequentemente constituem um dos principais gargalos da execução paralela em larga escala [Luu et al. 2015, Xie et al. 2020]. À medida que o número de nós cresce, requisições concorrentes ao

*Este projeto foi parcialmente financiado com recursos de bolsas de produtividade do CNPq (PQ-C: 304308/2025-0) e da FAPERJ (JCNE: E-26/204.481/2025). Recursos computacionais do SDumont alocados ao projeto wnnParIO

sistema de arquivos, contenção sobre adaptadores de rede e custos de coordenação entre processos passam a limitar a escalabilidade efetiva, mesmo quando há paralelismo computacional disponível. Para enfrentar esse desafio, o padrão *Message Passing Interface* (MPI) incorporou, a partir do MPI-2, a especificação MPI-IO, que provê primitivas para acesso paralelo a arquivos compartilhados [Message Passing Interface Forum 2025]. Combinada a sistemas de arquivos paralelos como o Lustre, a MPI-IO permite que múltiplos processos gravem simultaneamente em regiões distintas de um mesmo arquivo, distribuindo a carga de E/S entre vários *Object Storage Targets* (OSTs) e contornando os gargalos típicos de arquiteturas com escritor entralizado [OpenSFS and EOFS 2025].

Esse cenário é especialmente relevante para as ferramentas de planejamento da operação de sistemas elétricos. O algoritmo *Stochastic Dual Dynamic Programming* (SDDP) [Pereira and Pinto 1991] é amplamente usado no despacho hidrotérmico. No Brasil, o Operador Nacional do Sistema Elétrico (ONS) utiliza o SDDP no Programa Mensal da Operação (PMO). O algoritmo decompõe o problema em uma malha bidimensional de subproblemas indexados por estágio e cenário; a fase de simulação final — em que a política já convergida é avaliada em larga escala sobre um conjunto amplo de cenários — é naturalmente paralela, pois os cenários são mutuamente independentes e podem ser distribuídos entre processos MPI segundo o padrão *bag-of-tasks* [Cirne et al. 2003]. A transição da operação programada para resolução horária, motivada pela necessidade de capturar horários de ponta, vales e rampas associados à variabilidade das fontes intermitentes, multiplica em uma ordem de grandeza o volume de dados produzido por cenário e estágio [Conejo et al. 2010, Zakaria et al. 2020]. Nesse regime, a computação deixa de dominar exclusivamente o tempo de execução, que passa a sofrer forte influência das operações de E/S. A arquitetura atual do SDDP concentra todas essas operações em um único processo escritor. Cada processo, ao resolver um cenário, envia seu bloco de resultados ao escritor via `MPI_Send`. O escritor, então, serializa as gravações em disco. Essa estratégia introduz dois gargalos: o adaptador de rede do nó escritor, que recebe todas as transferências concorrentes, e o próprio caminho de escrita serializado em um único processo, que não aproveita o paralelismo do sistema de arquivos subjacente. A expansão dos ambientes computacionais utilizados no planejamento energético — de supercomputadores como o Santos Dumont (LNCC) à nuvem, como a Amazon Web Services (AWS) com FSx for Lustre — acentua esses gargalos, já que cada plataforma impõe características distintas de hardware, rede e sistema de arquivos.

Este artigo propõe e avalia uma arquitetura paralela de E/S para o SDDP, baseada em escritas independentes com MPI-IO sobre o sistema de arquivos Lustre, com o objetivo de substituir a estratégia atual com processo escritor único. O foco é a *fase de simulação final* do SDDP e, mais especificamente, a *escrita dos resultados* dessa fase, em que se cumula o maior volume de dados. Estão fora do escopo a fase iterativa de geração de cortes, a paralelização das leituras de entrada e o escalonamento de tarefas entre *ranks*, mantido conforme a estratégia *bag-of-tasks* existente. As principais contribuições são: (i) o diagnóstico dos gargalos da arquitetura atual de E/S do SDDP, com identificação do papel do processo escritor único; (ii) a proposta de uma arquitetura de E/S paralela baseada em MPI-IO sobre Lustre, com escritas independentes por processo em *offsets* previamente calculados, preservando o formato dos arquivos consumidos pela cadeia de softwares do planejamento; (iii) a instrumentação detalhada das duas implementações com métricas operacionalmente equivalentes; (iv) a avaliação experimental comparativa

em dois ambientes distintos — AWS com FSx for Lustre dedicado e Santos Dumont com Lustre compartilhado —, de 2 a 32 nós; e (v) a análise da influência do número de OSTs na escalabilidade da E/S paralela.

O restante deste artigo segue esta organização: a Seção 2 apresenta os conceitos fundamentais, apresentando os detalhes relevantes do SDDP, no contexto deste trabalho. A Seção 3 detalha nossa proposta de melhoria na distribuição de carga de E/S para o SDDP. A Seção 4 traz a avaliação e a análise experimental da abordagem aqui proposta, enquanto a Seção 5 discute brevemente os principais trabalhos sobre o tema encontrados na literatura. Finalmente, a Seção 6 fornece as considerações finais e delineia direções para trabalhos futuros.

2. Fundamentação

O MPI é o padrão dominante para troca de mensagens (`MPI_Send`, `MPI_Recv`, operações coletivas) e suporta E/S paralela via MPI-IO a partir do MPI-2 [Message Passing Interface Forum 2023, Message Passing Interface Forum 1997]. Centralizar resultados em um único escritor transforma a E/S em comunicação ponto a ponto, enquanto o MPI-IO delega a escrita aos processos, deslocando o gargalo para o sistema de arquivos. Em *bag-of-tasks*, o `MPI_Send` impõe custo de bloqueio ao remetente que escala com o volume e o número de processos concorrentes por um único receptor. Para E/S paralela, o MPI-IO oferece três abordagens: *escritas independentes* (`MPI_File_write_at`), simples e robustas para tarefas assíncronas sem exigência de participação coletiva [Cirne et al. 2003]; *escritas coletivas* (`MPI_File_write_all`), que sincronizam o comunicador permitindo otimizações por *two-phase I/O* e *data sieving*; e *variantes não bloqueantes* (`MPI_File_iwrite_at`), que sobrepõem computação e E/S sob a restrição de validade de *buffer*. Complementarmente, o Lustre separa metadados, gerenciados pelo *Metadata Server* (MDS), do fluxo de dados executado com os *Object Storage Servers* (OSSs) e *Object Storage Targets* (OSTs) [Braam 2002, OpenSFS and EOFS 2025]. O desempenho escala via *striping*, distribuindo blocos (*stripe size*) circularmente por múltiplos OSTs (*stripe count*), onde tamanhos maiores favorecem acessos sequenciais e menores beneficiam concorrência fina.

2.1. O algoritmo SDDP e sua arquitetura atual de E/S

O SDDP [Pereira and Pinto 1991] decompõe o planejamento hidrotérmico em subproblemas indexados por $(\text{estágio}, \text{cenário})$, resolvidos em duas fases: na *convergência*, iterações *forward/backward* constroem uma política representada por cortes de Benders; na *simulação final*, executada após a convergência, a política fixa é avaliada em larga escala sobre um conjunto amplo de cenários. Este trabalho concentra-se nessa segunda fase, em que os cenários são mutuamente independentes e caracterizam uma aplicação *bag-of-tasks* [Cirne et al. 2003]: a aplicação particiona os cenários entre os *ranks* MPI e os resolve em paralelo, de modo que o tempo total fica limitado pelo cenário mais lento. É essa natureza que torna a fase escalável, mas também desloca o gargalo da computação para a persistência dos resultados à medida que cresce o número de cenários. A granularidade temporal adotada define o volume produzido: a abordagem tradicional agrega cada

estágio em poucos *patamares* de carga, ao passo que a operação moderna requer resolução *horária* — foco deste trabalho —, em que cada estágio de um cenário gera dados proporcionais ao número de horas simuladas (p. ex. 720 registros por par $(\text{estágio}, \text{cenário})$).

Na arquitetura atual (Figura 1), apenas o *rank* 1 (processo escritor) realiza todas as operações de E/S, enquanto o *rank* 0 atua como distribuidor de tarefas. Cada nó, após resolver sua tarefa, envia os resultados por `MPI_Send`; o escritor os recebe por `MPI_Recv` e grava no sistema de arquivos. Independentemente do número de nós, um único processo concentra a E/S, o que gera dois gargalos: a saturação do adaptador de rede do nó escritor, que recebe todas as transferências concorrentes, e a serialização do fluxo de recepção e escrita. Um atraso do escritor obriga os nós trabalhadores a aguardar, aumentando o custo da execução paralela.

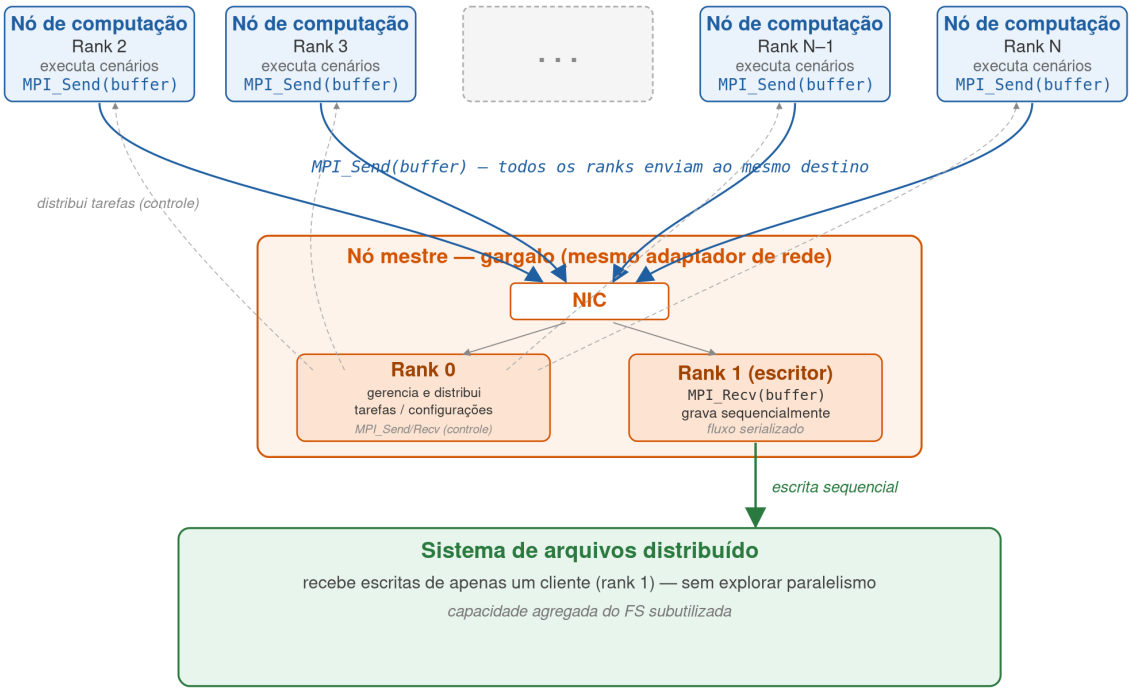


Figura 1. Arquitetura centralizada de E/S do SDDP (implementação atual): o *rank* 0 distribui tarefas, o *rank* 1 é o único escritor e concentra todas as operações de escrita, e os demais *ranks* executam os cenários.

3. Paralelização dos Mecanismos de E/S do SDDP

Nossa solução substitui o modelo atual por uma organização baseada em MPI-IO sobre o Lustre. Nessa abordagem, cada processo grava diretamente seus próprios resultados em regiões independentes do espaço de saída, evitando a concentração das operações de E/S em um único processo. O ponto central é explorar, simultaneamente, o paralelismo da aplicação MPI e o do sistema de arquivos: como os resultados de diferentes processos são independentes, a aplicação pode posicionar suas escritas previamente e executá-las em paralelo, enquanto o Lustre distribui os arquivos em *stripes* entre diferentes OSTs, atendendo a múltiplas escritas concorrentes e aumentando a banda agregada. A proposta reduz, assim, dois gargalos da arquitetura original: o de comunicação, pois

a aplicação deixa de enviar os dados ao processo escritor antes de persistí-los; e o de armazenamento, pois a escrita passa a ser distribuída entre os recursos do sistema de arquivos paralelo em vez de serializada em um único processo. Na arquitetura proposta (Figura 2), o *rank* 0 segue como distribuidor, enquanto os demais processos gravam seus resultados de forma independente sobre arquivos compartilhados, operando concorrentemente sobre o mesmo arquivo lógico em *offsets* distintos por meio das primitivas do MPI-IO [Message Passing Interface Forum 1997].

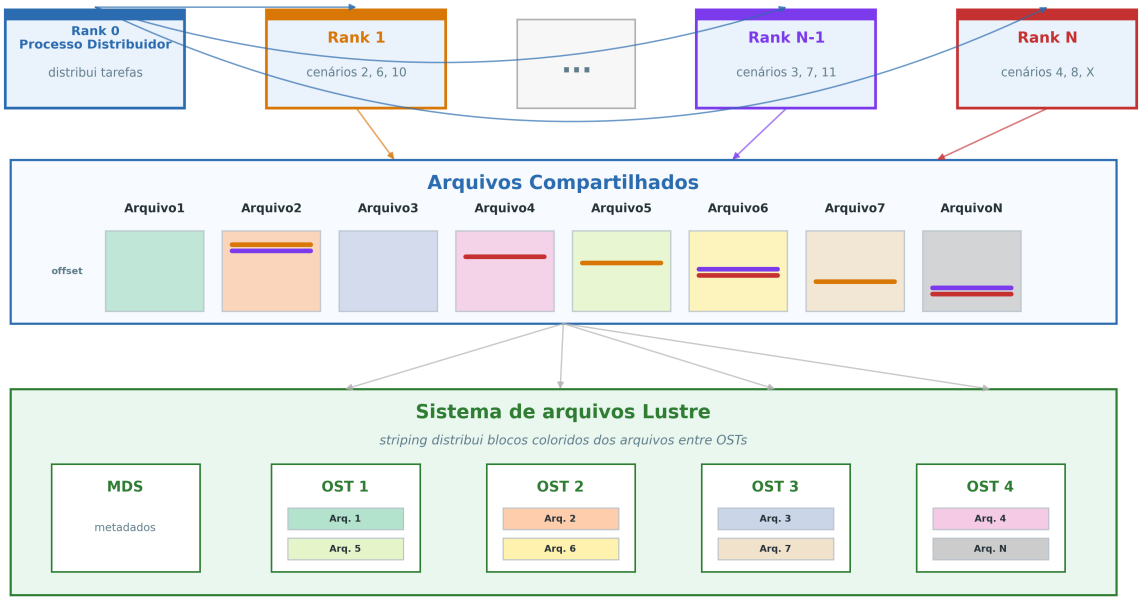


Figura 2. Arquitetura proposta com MPI-IO e Lustre: o *rank* 0 atua como distribuidor e os demais processos escrevem diretamente em arquivos compartilhados.

3.1. Modelo de execução

A Figura 3 ilustra a linha do tempo da execução segundo a arquitetura proposta, destacando a separação entre o processo distribuidor, a chamada coletiva `MPI_File_open` entre os *ranks* trabalhadores, as escritas independentes e a chamada `MPI_File_close`. A coordenação global fica restrita aos pontos de abertura e fechamento dos arquivos, enquanto as escritas realizadas após a abertura permanecem independentes. O *rank* 0 distribui as tarefas e, quando não está enviando uma nova tarefa, aguarda o recebimento das concluídas; por isso sua linha é representada integralmente como comunicação. Antes da primeira escrita, os trabalhadores participam de uma abertura coletiva dos arquivos (`MPI_File_open`), que atua como barreira; o distribuidor não participa dessa chamada.

Cada *rank* trabalhador executa o ciclo *bag-of-tasks*: recebe uma tarefa correspondente a um cenário, resolve a simulação e, ao término, efetua escritas independentes nos diversos arquivos de resultado, persistindo os blocos no formato proprietário do SDDP; concluída a persistência, solicita uma nova tarefa. Como o tempo de resolução de cada cenário varia, a distribuição dinâmica conduz a um desbalanceamento natural da carga de escrita entre o trabalhadores. Esse desbalanceamento, intrínseco ao padrão *bag-of-tasks*,

dificulta a adoção de escritas coletivas [Zhang et al. 2009]: chamadas coletivas exigiriam que todos os *ranks* participassem conjuntamente da mesma operação, forçando os processos mais rápidos a aguardar os mais lentos a cada ponto coletivo. Por essas razões, a estratégia de escritas independentes mostra-se mais aderente ao modelo de execução. Quando não há mais cenários a resolver, a aplicação realiza o fechamento coletivo dos arquivos (`MPI_File_close`).

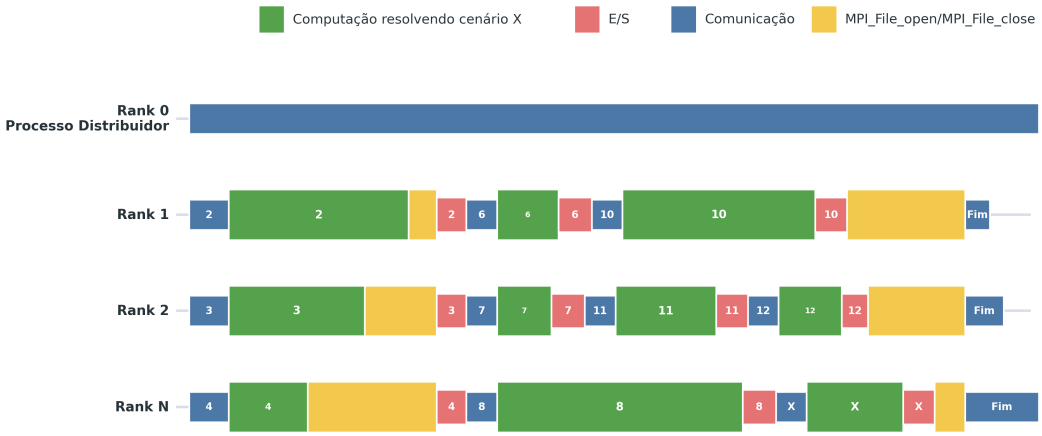


Figura 3. Linha de tempo da simulação final do SDDP com MPI-IO: os *ranks* trabalhadores executam cenários, sincronizam na abertura coletiva, realizam escritas independentes e sincronizam no fechamento.

4. Avaliações

A avaliação compara a implementação atual de E/S com a proposta, focando em tempo de execução, comunicação, largura de banda e bloqueio de E/S [Thakur et al. 1999, Carns et al. 2009] sobre a base PMO de outubro de 2023. Os experimentos empregaram dois ambientes: a nuvem **AWS** (com instâncias ENA e Amazon FSx for Lustre sob *Placement Group cluster*) [Amazon Web Services 2024b, Amazon Web Services 2024c] e o supercomputador **Santos Dumont (SDumont/LNCC)**, com InfiniBand EDR e MPICH 3.2 via *ch3* sobre IP (sem RDMA) [Petrini et al. 2003, Graham et al. 2005].

Para a simulação horária, executaram-se 1.536 cenários com três estágios mensais em ambos os ambientes, com cinco repetições por configuração (reportando média e desvio padrão). O mapeamento utilizou um *rank* por núcleo físico (sem *hyper-threading*), pois o consumo superior a 8 GB de RAM por *rank* esgotaria a capacidade de 384 GB por nó se dobrado. Trata-se de um cenário de *strong scaling* [Amdahl 1967], com tamanho de problema fixo e variação no número de nós.

A instrumentação via `MPI_Wtime` no módulo de persistência assegura comparação direta. A *banda agregada percebida pela aplicação* (B_w) expressa a eficiência de E/S para janelas de duração fixa Δ: é a razão entre o volume total gravado e o tempo de E/S do *rank* mais lento da janela ($B_w = \sum V_{i,w} / \max T^{\mathrm{E/S}}_{i,w}$), com valores médios entre janelas e repetições. O uso do processo mais lento no denominador reflete o comportamento síncrono da aplicação ao término de cada janela.

4.1. Caracterização do *workload* de escrita

O tamanho dos blocos é determinante para a E/S paralela. Blocos pequenos dominam em *frequência*: 1.801.728 escritas (80,1% do total) têm até 1 KB, enquanto as faixas intermediárias somam 152.082 (6,8%), os blocos de até 1 MB respondem por 267.264 (11,9%) e os de até 50 MB por 27.648 (1,2%). Em *volume*, porém, o quadro se inverte: praticamente todo o total escrito (cerca de 567 GB) concentra-se nas duas maiores faixas — aproximadamente 117 GB nos blocos de até 1 MB e 446 GB nos de até 50 MB. Esse contraste — escritas pequenas predominam em quantidade, grandes em bytes — decorre da organização dos 117 arquivos de resultado gerados: 104 no formato horário, em que os registros de um par $(\text{estágio}, \text{cenário})$ são acumulados em memória e gravados em uma única operação agregada, e 13 no formato diário, em que cada dia é gravado individualmente, respondendo pela maioria dos blocos pequenos. A alta frequência de operações de granularidade fina tende a aumentar a sobrecarga de metadados, enquanto o volume concentrado nos blocos maiores impõe a maior demanda sobre a largura de banda [Thakur et al. 1999, Carns et al. 2009].

4.2. Experimento AWS

O Experimento AWS utilizou 32 nós da instância r7i.12xlarge (Intel Xeon Sapphire Rapids 8488C a 3,2 GHz, 24 núcleos físicos, 384 GB DDR5, rede ENA de até 18,75 Gbps) [Amazon Web Services 2024b]. Empregou-se MPICH 3.2 e um Lustre v2.12 de 12 TB via Amazon FSx, com 1 MDS, 10 OSTs, *stripe count* 10 e *stripes* de 1 MB. Alocou-se um *rank* por núcleo físico, totalizando 24 *ranks* por nó [Intel Corporation 2023, Amazon Web Services 2024a].

Na **implementação atual** (Tabela 1, colunas *atual*), o tempo total cai com o aumento de nós até 16 nós (de 8.971 s em 2 nós para 1.874 s em 16 nós), mas volta a crescer para 2.013 s em 32 nós — regime saturado típico de *strong scaling* limitado por E/S. A banda degrada-se de 60,5 Gb/s em 2 nós para 3,4 Gb/s em 32 nós, e o percentual do tempo dedicado à E/S cresce de 0,34% para 49,51%, coerente com a serialização no nó escritor como ponto de contenção. Na **implementação MPI-IO** (Tabela 1), em que cada processo escreve diretamente no FSx o tempo total decresce monotonicamente até 1.274 s em 32 nós, o tempo de E/S permanece próximo ou abaixo de 1% a partir de 4 nós e a banda cresce de forma consistente até 380,3 Gb/s em 32 nós.

A comparação direta (Tabela 2) evidencia o ganho: a melhoria no tempo total é pequena em 2 nós, torna-se negativa em 4 nós, mas passa a positiva a partir de 8 nós, crescendo até 36,7% em 32 nós. A banda MPI-IO passa de ligeiramente inferior em 2 nós para 2,07$\times$ em 4 nós, 8,82$\times$ em 8 nós, 60,66$\times$ em 16 nós e 113,00$\times$ em 32 nós, e a redução do tempo de E/S chega a 99,1% em 32 nós.

A Figura 4a apresenta a banda agregada percebida, com curvas qualitativamente opostas. A implementação atual parte de 60,5 Gb/s em 2 nós — em pequena escala chega a superar a MPI-IO (57,0 Gb/s), pois o gargalo do escritor único ainda não saturou e o custo fixo da coordenação pesa relativamente — mas degrada-se mais de uma ordem de grandeza (18,9 Gb/s em 8 nós; 4,5 em 16; 3,4 em 32) à medida que a serialização no nó escritor se torna o ponto de contenção. A MPI-IO, ao contrário, cresce monotonicamente (106,0 Gb/s em 4 nós; 167,0 em 8; 274,2 em 16; 380,3 em 32), pois distribui as escritas

Tabela 1. Experimento AWS: desempenho das implementações atual e MPI-IO (banda percebida, tempo de E/S, percentual de E/S, tempo de `MPI_File_open`/`MPI_File_close`, tempo total e eficiência). Valores em média ± desvio padrão de cinco repetições, exceto % E/S e eficiência (pontuais); eficiência relativa à base de 2 nós; a coluna *open/close* aplica-se apenas à MPI-IO.

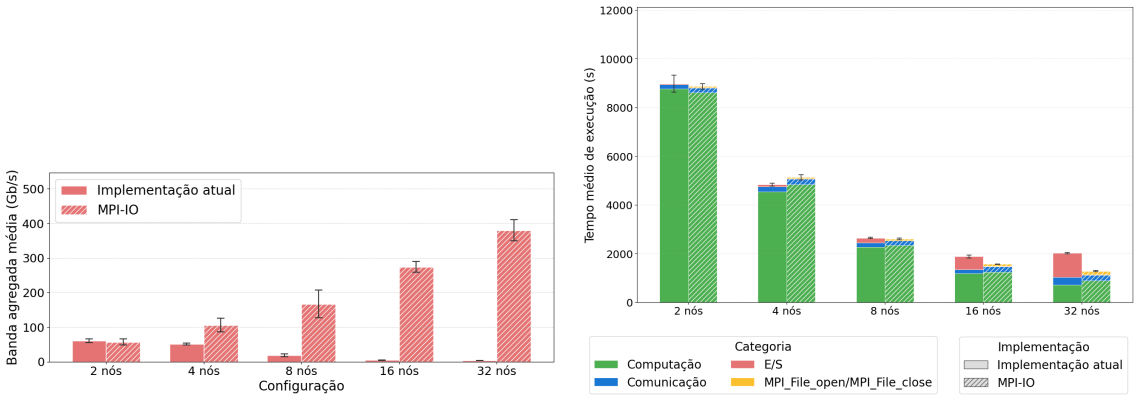
	Atual					MPI-IO					
Nós	Banda (Gb/s)	E/S (s)	% E/S	Total (s)	Efic.	Banda (Gb/s)	E/S (s)	% E/S	*open/ close* (s)	Total (s)	Efic.
2	60,5±5,2	30,9±13,2	0,3%	8 971±349	100%	57,0±8,5	19,0±5,6	0,2%	42,5±7,7	8 859±111	100%
4	51,3±2,8	78,2±27,8	1,6%	4 823±53	93%	106,0±19,4	12,8±3,5	0,3%	51,2±15,2	5 127±108	86%
8	18,9±3,8	201,7±53,3	7,6%	2 639±18	85%	167,0±40,2	10,8±6,3	0,4%	69,8±4,7	2 600±37	85%
16	4,5±0,3	527,9±53,1	28,2%	1 874±57	60%	274,2±15,3	7,5±1,8	0,5%	83,5±4,2	1 558±9	71%
32	3,4±0,1	996,6±167,0	49,5%	2 013±29	28%	380,3±30,7	9,0±2,5	0,7%	147,3±7,3	1 274±24	44%

Tabela 2. Experimento AWS: resumo comparativo atual × MPI-IO (melhorias calculadas em relação à implementação atual; valores negativos indicam piora).

Nós	Total atual (s)	Total MPI-IO (s)	Melhoria total	Ganho de banda	E/S atual (s)	Melhoria E/S
2	8 971	8 859	1,3%	0,94×	30,9	38,5%
4	4 823	5 127	−6,3%	2,07×	78,2	83,7%
8	2 639	2 600	1,5%	8,82×	201,7	94,7%
16	1 874	1 558	16,8%	60,66×	527,9	98,6%
32	2 013	1 274	36,7%	113,00×	996,6	99,1%

entre os processos e os múltiplos OSTs do FSx — um ganho sobre a implementação atual que passa de 2,07× em 4 nós a 8,82× em 8, 60,66× em 16 e 113,00× em 32 nós.

A Figura 4b decompõe o tempo de execução. Na implementação atual, a fração de E/S cresce de 7,6% em 8 nós para 28,2% em 16 e 49,5% em 32, fazendo o tempo total saturar em 16 nós e *piorar* em 32 nós (de 1.874 para 2.013 s): acrescentar nós deixa de compensar. Na MPI-IO, a E/S permanece abaixo de 1% a partir de 4 nós e o tempo total decresce monotonicamente até 1.274 s em 32 nós, o que corresponde a reduções de 1,5%, 16,8% e 36,7% em 8, 16 e 32 nós. A única parcela que cresce em termos relativos na MPI-IO é a das chamadas `MPI_File_open`/`MPI_File_close` (coluna *open/close* da Tabela 1), de 42,5 s (0,5%) em 2 nós a 147,3 s (11,6%) em 32. Esse custo é pago uma única vez por execução, na abertura e no fechamento dos arquivos (início e fim), e *independe do tamanho da carga*: não cresce com o número de cenários nem de estágios, dependendo apenas do número de nós que participam da operação coletiva. Por isso sua fração percentual aparece inflada, já que estes experimentos usam apenas três estágios mensais; mantido o mesmo número de nós, ao quadruplicar o horizonte (de 3 para 12 estágios) todas as componentes da decomposição crescem proporcionalmente enquanto esse custo permanece inalterado, tornando-se desprezível frente ao tempo total.



(a) Banda agregada percebida. **(b) Tempo de execução.**

Figura 4. Experimento AWS: (a) banda agregada percebida e (b) tempo de execução.

4.2.1. Impacto do número de OSTs

Para isolar o efeito da infraestrutura de armazenamento, comparou-se, na abordagem MPI-IO, configurações de 4 e 10 OSTs em 16 e 32 nós, mantidos constantes os demais parâmetros. A Figura 5 mostra que, em 16 nós, 10 OSTs atingem 274,2 Gb/s contra 49,4 Gb/s com 4 OSTs (ganho de 5,6×). Ao passar para 32 nós, as tendências se opõem: com 10 OSTs a banda cresce para 380,3 Gb/s, enquanto com 4 OSTs cai para 16,1 Gb/s — ganho de 23,6×. O tempo médio de envio nas classes de maior volume também se reduz drasticamente com mais OSTs (p. ex. de 309 ms para 13,8 ms nas mensagens de até 1 MB em 32 nós). Quando a concorrência entre processos aumenta, a configuração com menos OSTs limita a escalabilidade da E/S, indicando que o dimensionamento da camada de armazenamento deve ser tratado como parâmetro de projeto.

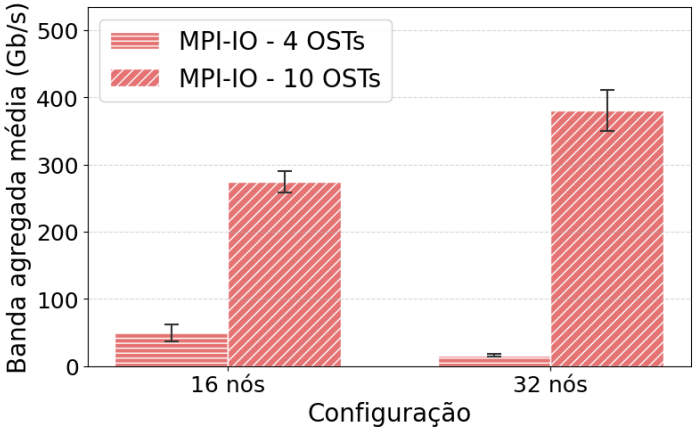


Figura 5. Banda agregada percebida na implementação MPI-IO para 4 e 10 OSTs em 16 e 32 nós (Experimento AWS).

4.3. Experimento SDumont

O Experimento SDumont utilizou até 32 nós (dois Intel Xeon Cascade Lake Gold 6252, 24 núcleos físicos, 384 GB, InfiniBand EDR de 100 Gb/s), MPICH 3.2 e um Lustre

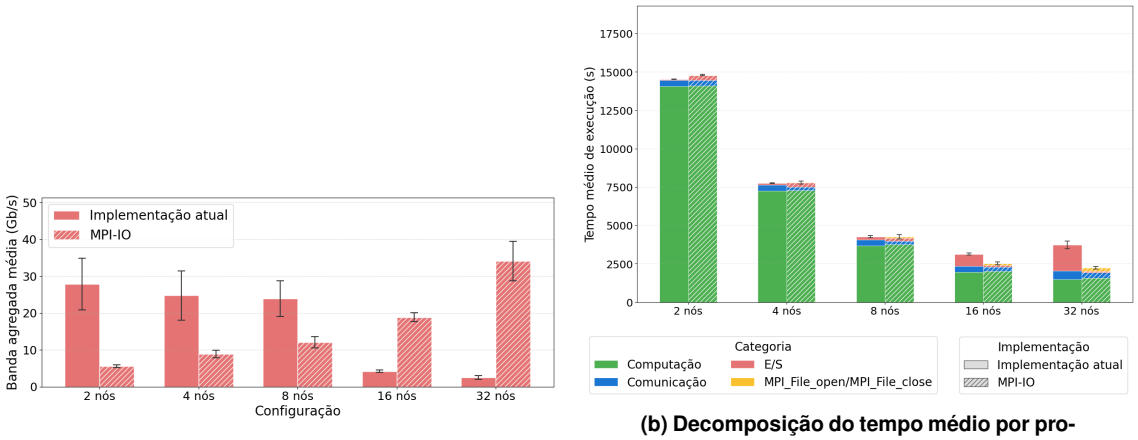
v2.12 (ClusterStor L300 da Cray/HPE) com 1 MDS e 6 OSTs, *stripe count* 6 e *stripes* de 1 MB. Diferentemente do FSx dedicado da AWS, o sistema de arquivos do SDumont é *compartilhado* com outras cargas de HPC, o que introduz variabilidade e amplifica a contenção nas maiores escalas — cenário mais desafiador para a escalabilidade de E/S.

O padrão observado na AWS se repete (Tabela 3). Na implementação atual, o tempo total decresce até 16 nós (3.130 s), mas piora em 32 nós (3.744 s), e a banda colapsa de 27,9 Gb/s em 2 nós para 2,5 Gb/s em 32 nós, com a parcela de E/S saltando para 45,83%. Na MPI-IO, o tempo total decresce em toda a faixa (até 2.233 s em 32 nós), a banda cresce consistentemente até 34,1 Gb/s e a E/S permanece abaixo de 5% em todas as configurações. A comparação direta mostra sobrecusto em pequena escala (–1,9% em 2 nós), mas ganhos crescentes a partir de 8 nós: 19,5% em 16 nós e 40,4% em 32 nós, com ganho de banda de 4,50× e 13,70× e redução do tempo de E/S de 87,3% e 96,3%, respectivamente.

Tabela 3. Experimento SDumont: desempenho das implementações atual e MPI-IO (banda percebida, tempo de E/S, tempo de `MPI_File_open`/`MPI_File_close`, tempo total e eficiência). Valores em média ± desvio padrão de cinco repetições, exceto eficiência (pontual); eficiência relativa à base de 2 nós; a coluna *open/close* aplica-se apenas à MPI-IO.

	Atual				MPI-IO				
Nós	**Banda (Gb/s)**	**E/S (s)**	**Total (s)**	**Efic.**	**Banda (Gb/s)**	**E/S (s)**	***open/ close* (s)**	**Total (s)**	**Efic.**
2	27,9±7,1	79,3±32,8	14 524±24	100%	5,5±0,4	321,2±85,2	23,2±3,0	14 796±46	100%
4	24,8±6,7	123,2±49,5	7 758±28	94%	8,9±1,0	271,5±50,0	32,3±1,6	7 800±93	95%
8	23,9±4,8	210,9±79,4	4 265±63	85%	12,1±1,5	194,7±34,4	79,3±21,9	4 251±142	87%
16	4,2±0,3	795,7±87,7	3 130±73	58%	18,9±1,2	100,9±21,1	126,5±28,3	2 519±101	73%
32	2,5±0,5	1 715,8±259,8	3 744±249	24%	34,1±5,4	64,1±27,0	230,0±8,9	2 233±87	41%

A Figura 6a confirma o contraste de *comportamento*: a implementação atual preserva banda elevada até 8 nós, mas colapsa a partir de 16 nós (entre 2,5 e 4,2 Gb/s), enquanto a MPI-IO cresce monotonicamente até 34,1 Gb/s em 32 nós. Diferentemente da AWS, porém, o SDumont é um supercomputador *compartilhado* em que centenas de *jobs* de diferentes usuários disputam simultaneamente os mesmos servidores de metadados e OSTs do Lustre. Essa concorrência externa impõe um teto de banda que atinge toda a aplicação: mesmo com a solução MPI-IO, a banda máxima no SDumont (34,1 Gb/s) fica cerca de uma ordem de grandeza *abaixo* da obtida na AWS com FSx dedicado (380,3 Gb/s). A escrita paralela preserva, portanto, a vantagem estrutural em relação ao escritor único. A Figura 6b decompõe o tempo médio por processo: na implementação atual, a parcela de E/S cresce marcadamente a partir de 16 nós, enquanto na MPI-IO ela permanece reduzida e o custo de `MPI_File_open`/`MPI_File_close` (coluna *open/close* da Tabela 3, de 23,2 s a 230,0 s de 2 a 32 nós) compõe a fração visível nas maiores configurações — pelo mesmo motivo já discutido para a AWS: um custo pago uma única vez, independente da carga e apenas inflado pelo horizonte reduzido destes experimentos. A queda de eficiência em 32 nós (24,2% na atual e 41,4% na MPI-IO) e a variabilidade refletem o caráter compartilhado do Lustre do SDumont; ainda assim, a vantagem estrutural da escrita paralela persiste em toda a faixa.



(a) Banda agregada percebida.

(b) Decomposição do tempo médio por processo.

Figura 6. Experimento SDumont: (a) banda agregada percebida e (b) decomposição do tempo médio por processo.

4.4. Síntese das avaliações

Os dois experimentos convergem para um diagnóstico comum, ainda que com magnitudes diferentes. Em pequena escala (2 e 4 nós), a implementação atual é competitiva ou ligeiramente superior, pois o gargalo do escritor único ainda não saturou e o custo fixo da camada MPI-IO pesa relativamente. Nas maiores escalas, o quadro se inverte — na AWS a partir de 8 nós, no SDumont a partir de 16 nós —, com a degradação da implementação atual refletindo a contenção sobre a serialização no nó escritor, enquanto a MPI-IO exibe perfil monotonicamente crescente de banda. Esse contraste qualitativo de comportamento — e não apenas o ganho pontual em uma escala — é o resultado mais robusto. A magnitude do benefício, porém, depende fortemente da infraestrutura: no FSx dedicado da AWS, a proposta atinge 380,3 Gb/s em 32 nós (113$\times$ sobre a atual) e reduz o tempo total em até 36,7%; no Lustre compartilhado do SDumont, chega a 34,1 Gb/s (13,7$\times$) e reduz o tempo total em até 40,4%, mesmo sob concorrência com outras cargas. A preservação da vantagem estrutural nesse regime sugere que a arquitetura é portátil entre ambientes heterogêneos.

Cabe ressalvar que, apesar dos ganhos de E/S, a eficiência paralela permanece bem abaixo do ideal linear (44% na AWS e 41% no SDumont em 32 nós). Isso é esperado no regime de *strong scaling*: com o problema de tamanho fixo, cada processo resolve menos cenários à medida que se adicionam nós, e custos quase constantes da computação — sobretudo os de acesso à memória — deixam de ser amortizados. Esse gargalo computacional, distinto do gargalo de E/S aqui tratado, foge ao escopo deste trabalho.

5. Trabalhos relacionados

Dickens e Logan [Dickens and Logan 2008] introduziram o uso de MPI-IO sobre Lustre, mostrando que, para padrões não-*interleaved* — em que cada processo escreve em região exclusiva —, escritas independentes evitam a contenção de *locks* e superam as coletivas; esse diagnóstico fundamenta a escolha de `MPI_File_write_at`

neste trabalho. Gropp *et al.* [Gropp et al. 2008] formalizaram que, nesse regime, `MPI_File_write_all` degenera para uma escrita simples acrescida de sincronização, tornando as coletivas estritamente piores — exatamente o padrão de acesso do SDDP. Liao *et al.* [Liao et al. 2007] demonstraram que escritas menores que uma unidade de *stripe* forçam múltiplos *locks* sequenciais e propuseram *two-stage write-behind buffering* alinhado às fronteiras de *stripe*; este trabalho não adota alinhamento explícito, mas parte das escritas do SDDP horário alinha-se incidentalmente ao *stripe* de 1 MB, o que ajuda a explicar o desempenho obtido. Dickens e Thakur [Dickens and Logan 2010] e Liao e Choudhary [Liao and Choudhary 2008] exploram estratégias adicionais de alinhamento e particionamento adaptativo para reduzir contenção no Lustre. Por fim, García-Carballeira *et al.* [Garcia-Carballeira et al. 2023] apresentam o *Expand Ad-Hoc*, um sistema de arquivos efêmero em espaço de usuário sobre o armazenamento local dos nós; compor a proposta com uma camada desse tipo poderia contornar a contenção externa observada no SDumont, com migração ao Lustre apenas no *stage-out*.

6. Conclusões

Este trabalho diagnosticou e resolveu o gargalo crítico de E/S na fase de simulação final do algoritmo SDDP, substituindo a arquitetura legada de um único processo escritor por uma abordagem paralela baseada em escritas independentes com MPI-IO no sistema de arquivos Lustre. A avaliação experimental conduzida em dois ambientes distintos — um sistema em nuvem com armazenamento dedicado (AWS/FSx) e um supercomputador tradicional com recursos compartilhados (LNCC/Santos Dumont) — evidenciou que a solução proposta elimina a contenção de rede e a serialização no nó mestre, garantindo escalabilidade em larga escala.

Mais do que os ganhos expressivos de desempenho e largura de banda observados em 32 nós, o resultado mais robusto deste estudo é o contraste qualitativo de comportamento: enquanto a estratégia centralizada colapsa devido à saturação do escritor, a arquitetura com MPI-IO exibe um perfil de escalabilidade monotonicamente crescente em ambas as infraestruturas. Isso demonstra não apenas a eficácia técnica da abordagem, mas também a sua portabilidade entre ecossistemas heterogêneos de computação de alto desempenho. Ademais, a análise do número de OSTs reforça que o dimensionamento da camada de armazenamento (*stripe count* e *stripe size*) deve ser tratado como parâmetro estrutural de projeto.

Como direções para trabalhos futuros, destacam-se a reestruturação do modelo de persistência dos arquivos diários para mitigar o overhead de E/S granular; a exploração de interconexões de altíssima velocidade (InfiniBand HDR/NDR, AWS EFA) com bufferização alinhada ao stripe do Lustre; a reavaliação de escritas coletivas com two-phase I/O; a investigação da piora pontual observada em 4 nós no ambiente AWS (Seção 4.2), onde o tempo total da implementação MPI-IO superou o da implementação atual apesar do ganho já mensurável de banda — indício de que fatores fora do módulo de E/S, possivelmente variações no tempo de computação entre repetições, dominaram esse ponto específico da curva; e a extensão da arquitetura à fase de geração de cortes do SDDP.

Referências

Amazon Web Services (2024a). Amazon EC2 R7i instances — Memory Optimized. https://aws.amazon.com/ec2/instance-types/r7i/.

Amazon Web Services (2024b). Elastic network adapter (ENA) — enhanced networking on Linux. Amazon EC2 User Guide.

Amazon Web Services (2024c). Placement groups for Amazon EC2 instances. Amazon EC2 User Guide.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18–20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA. Association for Computing Machinery.

Braam, P. J. (2002). The Lustre storage architecture. *Cluster File Systems Inc.*

Carns, P., Latham, R., Ross, R., Iskra, K., Lang, S., and Riley, K. (2009). 24/7 characterization of petascale I/O workloads. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE.

Cirne, W., Brasileiro, F., Sauvé, J., Andrade, N., Paranhos, D., Santos-Neto, E., and Medeiros, R. (2003). Grid computing for bag of tasks applications. In *Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government (I3E)*.

Conejo, A. J., Carrión, M., and Morales, J. M. (2010). *Decision Making Under Uncertainty in Electricity Markets*, volume 153 of *International Series in Operations Research & Management Science*. Springer, New York, NY.

Dickens, P. M. and Logan, J. (2008). Towards an understanding of the performance of MPI-IO in Lustre file systems. In *Proceedings of the IEEE International Conference on Cluster Computing*, CLUSTER '08, pages 254–263. IEEE.

Dickens, P. M. and Logan, J. (2010). A high performance implementation of MPI-IO for a Lustre file system environment. *Concurrency and Computation: Practice and Experience*, 22(11):1433–1449.

Garcia-Carballeira, F., Camarmas-Alonso, D., Calderon-Mateos, A., and Carretero, J. (2023). A new Ad-Hoc parallel file system for HPC environments based on the Expand parallel file system. In *Proceedings of the 22nd International Symposium on Parallel and Distributed Computing*, ISPDC '23, pages 69–76. IEEE.

Graham, R. L., Shipman, G. M., Barrett, B. W., Castain, R. H., Bosilca, G., and Lumsdaine, A. (2005). Open MPI: A high-performance, heterogeneous MPI. In *Proceedings of the 5th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*.

Gropp, W., Kimpe, D., Ross, R., Thakur, R., and Träff, J. L. (2008). Self-consistent MPI-IO performance requirements and expectations. In *Proceedings of the 15th European PVM/MPI Users' Group Meeting*, EuroPVM/MPI '08, pages 167–176. Springer.

Intel Corporation (2023). Intel Xeon scalable processors — 4th generation (Sapphire Rapids). https://www.intel.com/content/www/us/en/products/details/processors/xeon/scalable.html.

Kang, Q., Lee, S., Hou, K.-y., Ross, R., Agrawal, A., Choudhary, A., and Liao, W.-k. (2020). Improving MPI collective I/O for high volume non-contiguous requests with intra-node aggregation. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2682–2695.

Liao, W.-k., Ching, A., Coloma, K., Choudhary, A., and Kandemir, M. (2007). Improving MPI independent write performance using a two-stage write-behind buffering method. In *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, IPDPS '07. IEEE.

Liao, W.-k. and Choudhary, A. (2008). Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, SC '08. IEEE.

Luu, H., Winslett, M., Gropp, W., Ross, R., Carns, P., Harms, K., Prabhat, 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 '15, pages 33–44, New York, NY, USA. Association for Computing Machinery.

Message Passing Interface Forum (1997). MPI-2: Extensions to the message-passing interface. Specification.

Message Passing Interface Forum (2023). MPI: A message-passing interface standard, version 4.1. `https://www.mpi-forum.org/`.

Message Passing Interface Forum (2025). MPI: A message-passing interface standard, version 5.0. `https://www.mpi-forum.org/docs/`.

OpenSFS and EOFS (2025). *Lustre Software Release 2.x: Operations Manual*. `https://doc.lustre.org/lustre_manual.pdf`.

Pereira, M. V. F. and Pinto, L. M. V. G. (1991). Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, 52(1–3):359–375.

Petrini, F., Kerbyson, D. J., and Pakin, S. (2003). The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, page 55. ACM.

Thakur, R., Gropp, W., and Lusk, E. (1999). On implementing MPI-IO portably and with high performance. *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, pages 23–32.

Xie, B., Oral, S., Zimmer, C., Choi, J. Y., Dillow, D., Klasky, S., Lofstead, J. F., Podhorszki, N., and Chase, J. S. (2020). Characterizing output bottlenecks of a production supercomputer: Analysis and implications. *ACM Transactions on Storage*, 15(4):1–39.

Zakaria, A., Ismail, F. B., Lipu, M. S. H., and Hannan, M. A. (2020). Uncertainty models for stochastic optimization in renewable energy applications. *Renewable Energy*, 145:1543–1571.

Zhang, Z., Espinosa, A., Iskra, K., Raicu, I., Foster, I., and Wilde, M. (2009). Design and evaluation of a collective IO model for loosely coupled petascale programming. arXiv preprint.