UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
ENGENHARIA AEROESPACIAL

ARTHUR MIGUEL PEREIRA GABARDO

PROCESSAMENTO DE ALTO DESEMPENHO EM PROJETOS DE ENGENHARIA: IMPLANTAÇÃO, DESENVOLVIMENTO E AVALIAÇÃO DE UMA SOLUÇÃO HPC

Joinville
2024

ARTHUR MIGUEL PEREIRA GABARDO

PROCESSAMENTO DE ALTO DESEMPENHO EM PROJETOS DE ENGENHARIA: IMPLANTAÇÃO, DESENVOLVIMENTO E AVALIAÇÃO DE UMA SOLUÇÃO HPC

Trabalho apresentado como requisito para obtenção do título de Bacharel em Engenharia Aeroespacial, no Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina.

Orientador: Prof. Dr. Pablo A. Jaskowiak

Joinville
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Gabardo, Arthur Miguel Pereira
Processamento de Alto Desempenho em Projetos de Engenharia : Implantação, Desenvolvimento e Avaliação de uma Solução HPC / Arthur Miguel Pereira Gabardo ; orientador, Pablo Andretta Jaskowiak, 2024.
105 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Joinville, Graduação em Engenharia Aeroespacial, Joinville, 2024.

Inclui referências.

1. Engenharia Aeroespacial. 2. Computação de Alto Desempenho. 3. programação paralela. 4. otimização multidisciplinar. 5. algoritmos genéticos. I. Jaskowiak, Pablo Andretta. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Aeroespacial. III. Título.

ARTHUR MIGUEL PEREIRA GABARDO

PROCESSAMENTO DE ALTO DESEMPENHO EM PROJETOS DE ENGENHARIA: IMPLANTAÇÃO, DESENVOLVIMENTO E AVALIAÇÃO DE UMA SOLUÇÃO HPC

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Engenharia Aeroespacial, no Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina.

Joinville (SC), 10 de Dezembro de 2024.

Banca Examinadora:

Prof. Dr. Pablo Andretta Jaskowiak
Orientador/Presidente
UFSC

Prof. Dr. Eng. Thiago Pontin Tancredi
Membro(a)
UFSC

Prof. Dr. Eng. Lucas Weihmann
Membro(a)
UFSC

A Alice, Anne e Rafael (in memoriam).

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha profunda gratidão aos meus pais, Rafael e Anne, e à minha avó Edit, cujo apoio incondicional e investimento na minha educação foram fundamentais para todas as minhas conquistas. Vocês são a base de tudo, e, sem o amor e dedicação de vocês, nada disso seria possível.

À minha querida irmã Alice, por sua alegria que torna meus dias mais especiais.

À minha amada Vitória, por ser minha companheira incansável durante toda essa jornada. Sua presença tornou cada desafio mais leve e cada conquista mais doce.

Aos meus Professores, em especial ao meu orientador e amigo, Pablo, pelos importantes ensinamentos, pela confiança depositada em mim e por cada oportunidade que me fez crescer. Também agradeço imensamente ao Prof. Pontin pelos desafios propostos e por propiciar os meios e a liberdade para o desenvolvimento deste trabalho.

Aos meus amigos de graduação e colegas de laboratório, por participarem e ajudarem na construção da minha história. Nossos momentos juntos foram muito valiosos.

À Petrobras S.A. pelo apoio financeiro.

“ *sudo rm -rf /** ”

Gabardo, Arthur (2023)

RESUMO

A crescente utilização de modelos computacionais, simulações e otimizações tem propiciado avanços significativos na investigação científica e em projetos de engenharia. Entretanto, a complexidade e custo computacional destes modelos, associados a grandes bases de dados, tem impulsionado a busca por infraestruturas de Processamento e Computação de Alto Desempenho (HPC), necessárias para a execução destas tarefas. Apesar de seu potencial para transformar práticas de engenharia, a implementação de infraestruturas HPC enfrenta desafios, como os altos custos de aquisição e especialização técnica demandada para manutenção e desenvolvimento destes sistemas. Diante desse cenário, a utilização de *Clusters Commodity*, compostos por hardware de baixo custo e fácil acesso no mercado, apresenta-se como uma alternativa estratégica para tornar essas tecnologias mais acessíveis. Este trabalho tem como objetivo detalhar o processo de implantação do Cluster Aqua, utilizando a infraestrutura preexistente em laboratório com recursos heterogêneos. O cluster foi posteriormente integrado ao sistema Synapse, voltado para a otimização de sistemas de engenharia, por meio do desenvolvimento de um *middleware* que viabiliza a execução distribuída de análises. A metodologia abrangeu a configuração do *hardware* e de uma pilha de *softwares*, seguido pela realização de testes de desempenho utilizando *benchmarks*, bem como a aplicação prática do cluster em experimentos computacionais utilizando o *middleware* proposto. Os resultados obtidos permitiram avaliar aspectos de desempenho, escalabilidade e viabilidade da solução, fornecendo subsídios para validar sua aplicabilidade em projetos de engenharia, bem como a ampliação das capacidades computacionais do laboratório com a incorporação da infraestrutura HPC implantada.

Palavra-chave: Computação de Alto Desempenho; programação paralela; otimização multidisciplinar; algoritmos genéticos.

ABSTRACT

The increasing integration of computational models, simulations, and optimizations has led to significant advancements in scientific research and engineering projects. However, the complexity and computational cost of these models, coupled with large datasets, have driven the search for High-Performance Computing (HPC) infrastructures, essential for executing these tasks. Despite its potential to transform engineering practices, the implementation of HPC infrastructures faces challenges, such as high acquisition costs and the technical specialization required for the maintenance and development of these systems. In this context, the use of commodity clusters, composed of low-cost hardware that is readily available on the market, emerges as a strategic alternative to make these technologies more accessible. This paper aims to report the process of deploying the Aqua Cluster, using pre-existing laboratory infrastructure with heterogeneous resources, and its integration into the Synapse software, focused on optimizing engineering systems, through the development of a middleware that enables the distributed execution of analyses. The methodology included configuring the hardware and software stack, followed by performance testing using benchmarks, as well as the practical application of the cluster in computational experiments using the proposed middleware. The results obtained allowed for the evaluation of performance, scalability, and viability aspects of the solution, providing support to validate its applicability in engineering projects, as well as enhancing the computational capabilities of the laboratory with the incorporation of the deployed HPC infrastructure.

Keywords: *High-Performance Computing; parallel programming; multidisciplinary optimization; genetic algorithms*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALU	Unidade Aritmética e Lógica
API	Interface de Programação de Aplicações
BLAS	Basic Linear Algebra Subprograms
BMC	Baseboard Management Controller
COTS	Commercial-off-the-shelf
CPU	Unidade Central de Processamento
CTJ	Centro Tecnológico de Joinville
CU	Unidade de Controle
CUDA	Arquitetura de Dispositivo de Computação Unificada
DHCP	Protocolo de Configuração Dinâmica de Host
DoE	Planejamento de Experimentos
EDVAC	Electronic Discrete Variable Automatic Computer
EPEL	Extra Packages for Enterprise Linux
FMA	Fused Multiply-add Operation
FSB	Barramento Frontal
GA	Algoritmo Genético
GPU	Unidade Gráfica de Processamento
HA	Cluster de Alta Disponibilidade
HDFS	Hadoop File System
HPC	Computação de Alto Desempenho
HPCC	Cluster de Computação de Alto Desempenho
HPL	High Performance LINPACK
I/O	Entradas e Saídas
IP	Protocolo da Internet
LAN	Rede de Área Local
MAC	Controle de Acesso ao Meio
MDO	Otimização Multidisciplinar de Projetos
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MOBO	Placa-Mãe
MPI	Interface de Passagem de Mensagens
NFS	Network File System
NSO	Núcleo de Simulação e Otimização de Sistemas Dinâmicos
NUMA	Acesso Não-Uniforme de Memória
OpenHPC	Open High Performance Computing
OpenMP	Open Multi-Processing

OS	Sistema Operacional
OSI	Open Systems Interconnection
PXE	Ambiente de Pré-execução
RAM	Memória de Acesso Aleatório
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SLURM	Simple Linux Utility for Resource Management
SMB	Server Message Block
SMP	Multiprocessamento Simétrico
SmS	Synapse meets Slurm
SSH	Secure Shell
TCP	Protocolo de Controle de Transmissão
UFSC	Universidade Federal de Santa Catarina
UMA	Acesso Uniforme de Memória
USL	Lei Universal da Escalabilidade
VLAN	Rede de Área Local Virtual
WAN	Rede de Área Ampla
xCAT	Extreme Cloud Administration Toolkit

LISTA DE SÍMBOLOS

R_{peak}	Desempenho Máximo Teórico
R_{max}	Desempenho Máximo Real
p	Número de Processos
T_1	Tempo de Execução Serial
T_p	Tempo de Execução Paralela
σ	Parcela Sequencial do Programa
κ	Limite de Coerência do Programa
η_{sist}	Eficiência do Sistema
S_p	Ganho de Desempenho
ε	Escalabilidade do Programa
p^*	Quantidade Ótima de Processos Paralelos
α	Fator de Segurança de Uso de Memória
$f(\vec{x})$	Função Objetivo
$\vec{x}$	Vetor de Parâmetros
C	Conjunto de Restrições

SUMÁRIO

1 INTRODUÇÃO

A investigação de fenômenos em ciência e engenharia compreende a formulação de teorias e modelos que os descrevam. Essas inquirições são tradicionalmente realizadas de maneira analítica ou experimental. Entretanto, nas últimas décadas, avanços tecnológicos proporcionaram melhora no desempenho e eficiência de microprocessadores fazendo com que cada vez mais otimizações, modelos e simulações computacionais sejam viáveis e aceitos como instrumentos de estudo (Post; Kendall; Lucas, 2006).

Modelos e simulações computacionais são amplamente utilizados em diversas áreas, como previsão climática, astrofísica, combustão, dinâmica dos fluidos e resistência dos materiais (Nagel *et al.*, 2023; Nagel *et al.*, 2022). Esses modelos computacionais não são substitutos de experimentos ou da teoria, mas oferecem a vantagem de tratar problemas complexos, de maneira rápida e com baixo custo (Tabela 1), fornecendo indicativos de quais experimentos são necessários e auxiliando na validação de novas teorias (Rüde *et al.*, 2018).

Tabela 1 – Comparação entre diferentes abordagens de investigação científica

Abordagem	**Vantagens**	**Desvantagens**
Experimental	• Mais realista	• Necessidade de equipamentos • Problemas de escala • Dificuldades de medição • Custos operacionais
Teórica	• Generalista • Fórmula fechada	• Restrito a problemas simples • Muito restrito à linearidade
Computacional	• Tratamentos de problemas de alta complexidade • Sem restrição de linearidade	• Erros de truncamento • Conhecimento de parâmetros do problema *a priori* • Custo computacional

Fonte: Adaptado de Anderson; Tannehill; Pletcher (2016, p. 9).

Além disso, a crescente importância de modelos de alta fidelidade em ciência e engenharia, associados a bases de dados massivas, impõe a necessidade de arquiteturas computacionais paralelas capazes de realizar operações simultâneas. É o caso dos sistemas de Computação de Alto Desempenho (*High Performance Computing* – HPC), máquinas que utilizam diversas unidades de processamento, núcleos ou *cores* de maneira cooperativa para solução de um problema comum (Hager; Wellein, 2019).

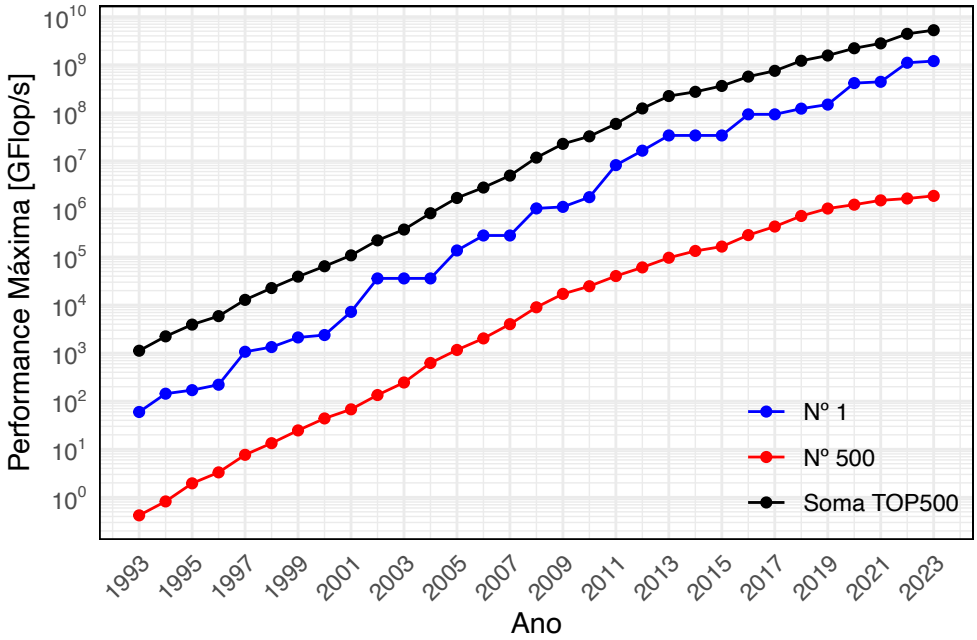
Portanto, Computadores de Alto Desempenho se tornaram uma ferramenta de amplo uso, tanto na academia quanto na indústria. Ademais, tendo em vista que o desempenho de sistemas computacionais modernos não depende apenas da densidade de

transistores nos microprocessadores, mas principalmente, da quantidade de operações simultâneas, nessas aplicações, é comum o agrupamento de vários computadores no que é conhecido como Clusters de Computação de Alto Desempenho (*High Performance Computing Clusters* – HPCCs) ou supercomputadores[1].

A diminuição dos custos e a inovação na fabricação de componentes como microprocessadores e Memórias de Acesso Aleatório (*Random Access Memories* – RAM), bem como o desenvolvimento de diferentes infraestruturas e protocolos de rede, possibilitaram a disseminação dos HPCCs em diferentes segmentos, incluindo o uso de *Commercials-off-the-shelf* (COTS) na construção de *Commodity Clusters*. Como resultado, em um curto período de tempo, supercomputadores tiveram um aumento na ordem de 10^{11} vezes em poder de processamento (Post *et al.*, 2006).

Anualmente, a organização TOP500 publica um ranking dos 500 melhores sistemas de alto desempenho considerando o *benchmark* LINPACK[2]. A Figura 1 apresenta a evolução de HPCCs nas últimas três décadas com base nos resultados deste *benchmark*. No início dos anos 1990, os melhores sistemas disponíveis tinham performance máxima de algumas dezenas de GFlops/s, comparáveis ao desempenho de processadores de uso pessoal modernos. Em contraste, no ano de 2023, o melhor supercomputador do ranking foi capaz de realizar $1{,}2 \times 10^9$ bilhões de operações por segundo (TOP500, 2023).

Figura 1 – Evolução na performance de supercomputadores nas últimas três décadas



Fonte: Adaptado de TOP500 (2023).

[1] Neste trabalho, os termos supercomputador, cluster e sistema HPC são usados de maneira intercambiável.

[2] LINPACK é um *benchmark* que mede a performance de um sistema computacional com base na quantidade de operações de ponto flutuante realizadas por unidade de tempo [Flops/s] na solução de um sistema denso de equações lineares (Dongarra, 1987).

Segundo Post *et al.* (2006), a complexidade, a escala e a heterogeneidade de sistemas HPCC modernos e suas aplicações, resultam em desafios em seu desenvolvimento, implantação e validação, que podem ser resumidos da seguinte forma:

- **Desafio de Performance**: projetar e integrar computadores de alto desempenho;
- **Desafio de Programação**: escrever códigos para execução paralela em computadores complexos e heterogêneos;
- **Desafio de Preditividade**: desenvolver modelos de sistemas complexos com resultados preditivos e reprodutíveis;
- **Desafio de Desenvolvimento**: apoiar e financiar iniciativas para o desenvolvimento de ciência e engenharia computacional.

Devido à crescente complexidade na modelagem computacional de sistemas de engenharia e problemas científicos, a implantação de Computadores de Alto Desempenho se mostra um desafio necessário. O presente trabalho aborda os desafios de performance e programação. Para isto, um Cluster de Alto Desempenho foi implementado em âmbito de *hardware* e *software* nas instalações do Núcleo de Simulação e Otimização de Sistemas Dinâmicos (NSO), do Centro Tecnológico de Joinville (CTJ), da Universidade Federal de Santa Catarina (UFSC), seguindo as receitas e utilizando os pacotes disponíveis no repositório da comunidade OpenHPC (2018).

A programação paralela foi usada para realizar a busca concorrente de soluções em algoritmos populacionais, que, segundo Tancredi (2009), apresentam ganhos de desempenho consideráveis quando processados de maneira distribuída. A simulação dos indivíduos deve ser feita em lotes e um algoritmo para balanceamento de carga e gerenciamento de filas foi proposto. Com os resultados deste trabalho busca-se avaliar de maneira quantitativa, por meio de *benchmarks* e medidas de *speedup*, as mudanças em eficiência dos algoritmos, tempo de processamento e custo computacional, propiciadas por paradigmas de processamento distribuído e paralelo nas simulações e otimizações de diferentes projetos de engenharia.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

De maneira geral, este trabalho concentra-se em dois objetivos centrais:

- Implementar uma arquitetura de *software* e *hardware* de um Cluster de Computação de Alto Desempenho (*High Performance Computing Cluster* – HPCC) para projetos de engenharia;
- Desenvolver um *middleware* para o processamento paralelo e distribuído de algoritmos populacionais, integrando o *software* Synapse a HPCCs.

1.1.2 Objetivos Específicos

A fim de concretizar os objetivos gerais, este trabalho define os seguintes objetivos específicos:

- Realizar a especificação detalhada da infraestrutura de processamento requerida e disponível para a implantação de um cluster de alto desempenho no NSO;
- Configurar a topologia de rede adequada para o cluster, garantindo eficiência e escalabilidade;
- Instalar e configurar os *softwares* e gerenciadores necessários para assegurar o pleno funcionamento do cluster;
- Desenvolver algoritmos para processamento paralelo e balanceamento de carga de processos de otimização;
- Avaliar a eficiência dos algoritmos implementados, comparando o desempenho, o tempo de processamento e o custo computacional entre abordagens tradicionais (*i.e.*, sequenciais) e soluções de alto desempenho.

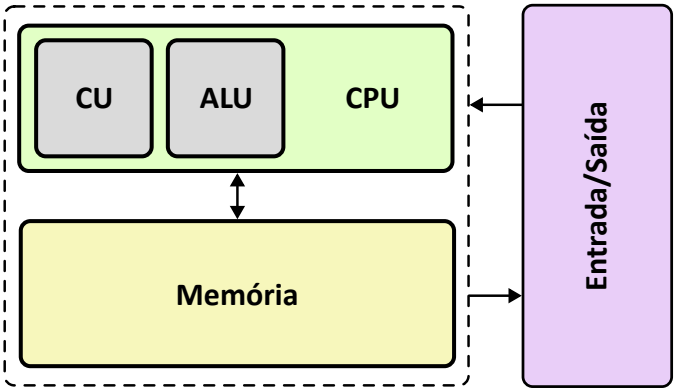
1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado em seis capítulos, incluindo esta Introdução. A seguir, apresenta-se uma visão geral do conteúdo abordado em cada um deles. O Capítulo 2 discute os fundamentos das arquiteturas de computação de alto desempenho, abrangendo conceitos sobre processamento paralelo e distribuído, topologias e protocolos de redes e pilhas de *software* utilizadas no gerenciamento desses sistemas. O objetivo principal é contextualizar a base teórica necessária para compreender os tópicos subsequentes. O Capítulo 3 foca em programação paralela, detalhando os principais modelos de paralelismo e padrões de programação paralela – incluindo estratégias baseadas em *multithreading*, passagem de mensagens e *MapReduce*. Adicionalmente, apresenta-se uma análise de métricas de desempenho fundamentais, como eficiência, escalabilidade e tolerância a falhas. No Capítulo 4, é descrita a implementação do cluster Aqua nas instalações do NSO, incluindo a especificação da infraestrutura de *hardware* e a configuração de *software*. Este capítulo também aborda os resultados obtidos por meio de *benchmarks*, analisando o desempenho da solução proposta. O Capítulo 5 apresenta o desenvolvimento de um *middleware* voltado para a paralelização de algoritmos de otimização populacionais. Essa ferramenta visa integrar clusters gerenciados pelo SLURM ao *software* Synapse, permitindo a execução eficiente de simulações distribuídas. Por fim, o Capítulo 6 reúne os comentários e as considerações finais deste trabalho, oferecendo uma síntese dos principais resultados e discutindo as contribuições alcançadas. Além disso, são sugeridas direções para futuros estudos e melhorias relacionadas ao uso de computação de alto desempenho em projetos de engenharia.

2 ARQUITETURAS DE COMPUTAÇÃO DE ALTO DESEMPENHO

Desde o desenvolvimento do *Electronic Discrete Variable Automatic Computer* (EDVAC) por John von Neumann (1945), a arquitetura básica mais implementada em computadores digitais permaneceu essencialmente inalterada[1]. Conhecida como Computador com Programa Armazenado, Modelo de von Neumann ou Arquitetura de Princeton (Figura 2), essa abordagem permite que programas e dados sejam armazenados no mesmo espaço de memória. A Unidade de Controle (*Control Unit* – CU) acessa esses dados e programas, enquanto a Unidade Aritmética e Lógica (*Arithmetic Logic Unit* – ALU) - ambas parte da Unidade Central de Processamento (*Central Processing Unit* – CPU) - realiza as operações necessárias, permitindo a manipulação de informações por meio das funções básicas de Entradas e Saídas (*Inputs and Outputs* – I/O) fornecidas ao usuário.

Figura 2 – Arquitetura tradicional de von Neumann



Fonte: Autor (2024).

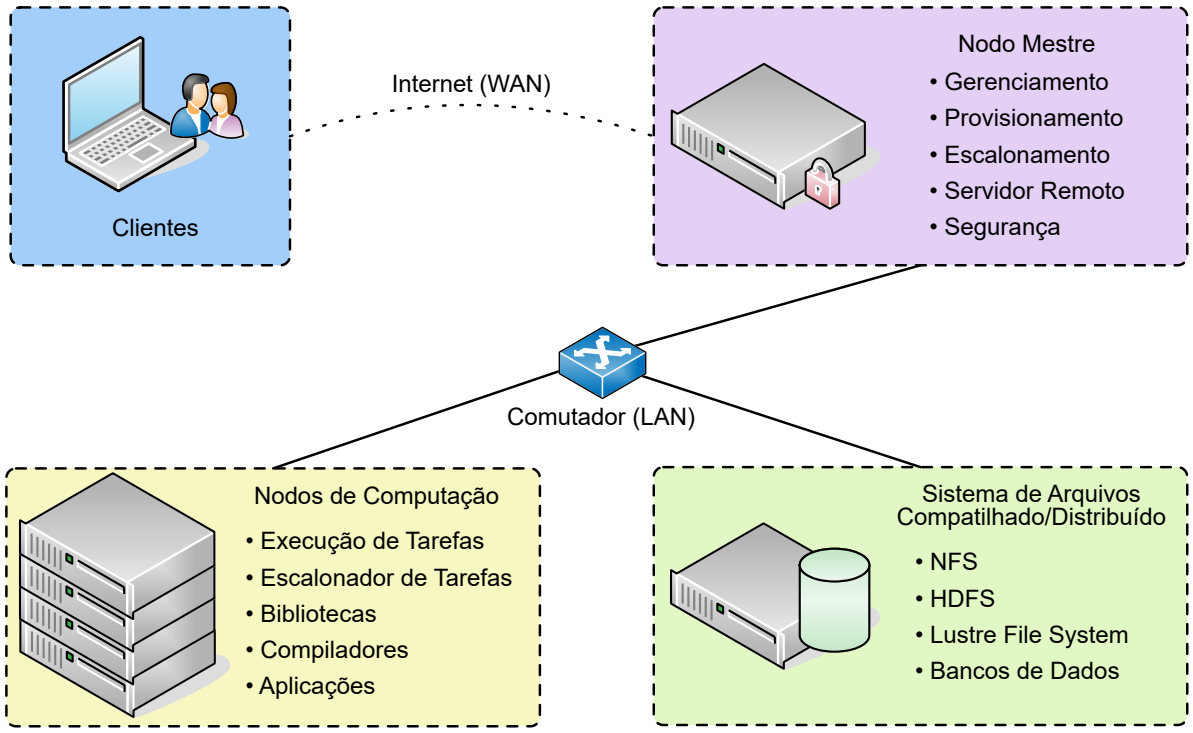
Atualmente, o conceito de arquitetura de computadores pode ser compreendido de maneira mais ampla, não se limitando apenas ao modo com que os dados armazenados em memória são manipulados pela CPU, mas abarcando os aspectos gerais tanto de *software* quanto *hardware* que permitem o funcionamento e usabilidade de sistemas como um todo. Essa definição mais abrangente inclui os diferentes tipos de processadores e seus conjuntos de instruções, as estruturas e protocolos utilizados para comunicação entre computadores, bem como sistemas operacionais, pilhas de *softwares* básicos e a estrutura e organização de arquivos em disco.

No contexto de Computação de Alto Desempenho (*High Performance Computing* – HPC), o estudo e aperfeiçoamento de diferentes arquiteturas tem promovido o aumento

[1] A Arquitetura de Harvard - uma modificação do modelo de von Neumann - é amplamente utilizada no processamento digital de sinais e sistemas embarcados de tempo-real. Diferencia-se pelo uso de memórias separadas para o conjunto de instruções e dados, evitando o gargalo de von Neumann. Tal arquitetura foge do escopo deste trabalho e para mais detalhes recomenda-se a leitura de Noergaard (2012).

do desempenho de computadores, viabilizando seu uso na simulação e otimização de modelos complexos (como os de engenharia) e no processamento de conjuntos de dados massivos. Os avanços em sistemas e protocolos de rede permitiram o uso distribuído desses recursos computacionais de maneira eficiente, dando origem aos HPCCs como na Figura 3. Estes sistemas consistem no agrupamento de múltiplas máquinas conectadas por uma rede, aumentando consideravelmente os recursos disponíveis para solução de um problema comum. HPCCs tem se tornado cada vez mais populares pela possibilidade de uso de COTS - computadores pessoais/workstations - na construção de pequenos sistemas HPC denominados de *Commodity Clusters*.

Figura 3 – Arquitetura de um Cluster de Computação de Alto Desempenho (HPCC)



Fonte: Autor (2024).

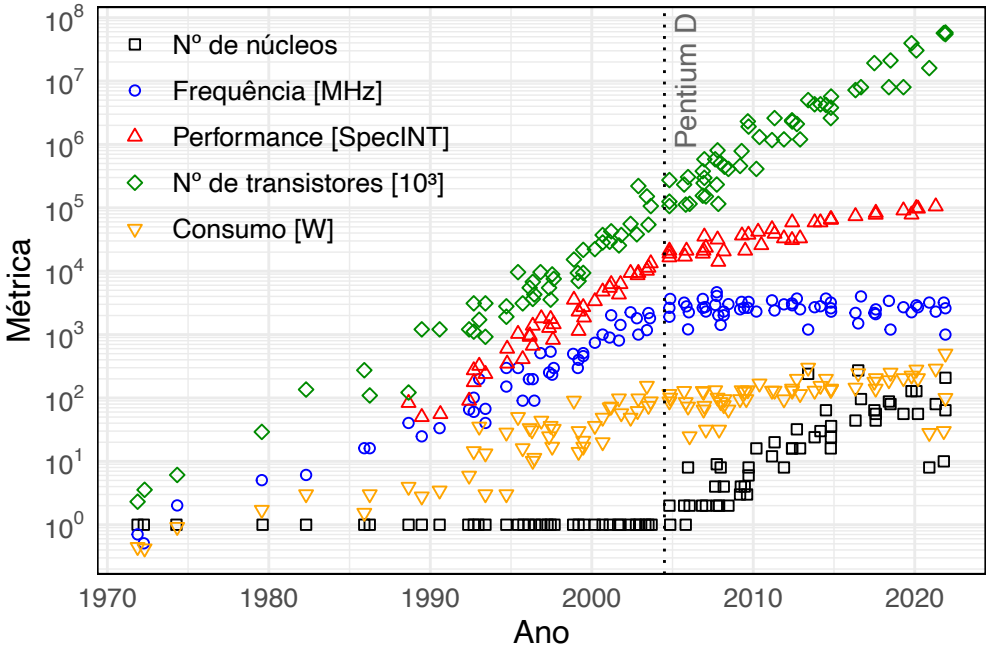
Este capítulo apresenta os conceitos fundamentais de sistemas de computação de alto desempenho, como os paradigmas de processamento paralelo/distribuído e compartilhamento de memória, redes de dados e seus protocolos, e pilhas de tecnologias focadas no gerenciamento de sistemas HPC.

2.1 PROCESSAMENTO PARALELO E DISTRIBUÍDO

Na década de 1960, Gordon Moore, co-fundador da Intel Corporation, por meio de evidências empíricas, conjecturou que o número de transistores (*i.e.*, os dispositivos eletrônicos que realizam operações lógicas nas CPUs) dobrariam a cada dois anos (Moore, 1965). A indústria tem se empenhado para cumprir de maneira rigorosa a previsão de Moore desde então. A Figura 4 agrega informações relativas ao avanço no desempenho dos

microprocessadores nos últimos 50 anos, e demonstra o avanço constante na quantidade de transistores integrados aos chips.

Figura 4 – Dados de tendências de microprocessadores nos últimos 50 anos



Fonte: Adaptado de Rupp (2022).

Inicialmente, a estratégia adotada foi a de redução no tamanho dos transistores e aumento da frequência dos microprocessadores. Entretanto, esta abordagem culminou com o aumento do consumo energético e temperaturas além das suportadas pelos componentes, impondo um limite na frequência e alterando o paradigma vigente na produção de CPUs. Em 2004, a Intel Corporation, com o Pentium D Dual Core, iniciou uma tendência que foi sucedida por outros na indústria, firmando um conceito que estava em estudo há cerca de 40 anos, os processadores multinúcleo (Mishra *et al.*, 2011).

Os processadores multinúcleo foram capazes de manter a meta de dobrar o número de transistores a cada dois anos, aumentando o desempenho sem elevar excessivamente as frequências ou o consumo de energia. Estes processadores contam com um ou mais núcleos de processamento, *i.e.*, unidades individuais capazes de realizar tarefas simultaneamente. No entanto, os núcleos não somam diretamente suas capacidades de processamento mas dividem as tarefas entre si. Para tirar proveito de múltiplos núcleos, programas devem ser desenvolvidos de maneira particular para execução paralela (Patterson *et al.*, 2011).

Tradicionalmente, programas computacionais são escritos para processamento sequencial, ou seja, o problema específico é particionado em uma série de instruções discretas, que são executadas uma após a outra em um único processador (Barney; Frederick, 2023). Em contrapartida, em processadores e programas paralelos/distribuídos, o

problema em questão é subdividido em diferentes partes que podem ser solucionadas de maneira simultânea. As instruções contidas nessas subdivisões são então executadas de maneira sequencial nos múltiplos núcleos e posteriormente tem seus resultados agregados (Trobec *et al.*, 2018).

Seguindo a taxonomia definida por Flynn (1966), é possível classificar os paradigmas de computação paralela em quatro categorias distintas, com base no fluxo de controle e dados simultâneos de uma determinada máquina:

- **Single Instruction Single Data (SISD)**: apenas uma única instrução pode ser executada sobre um único fluxo de dados por ciclo de processamento. Este paradigma corresponde aos primeiros tipos de computadores, atualmente em desuso;
- **Single Instruction Multiple Data (SIMD)**: uma instrução é realizada simultaneamente em diversos fluxos de dados distintos. Utilizado para problemas especializados, como as operações realizadas por Unidades Gráficas de Processamento (*Graphics Processing Unit* – GPUs) e aceleradores científicos (*e.g.*, AMD Alveo[2] e NVIDIA Jetson[3]);
- **Multiple Instruction Single Data (MISD)**: múltiplas instruções são realizadas no mesmo fluxo de dados. Este paradigma possui pouca aplicação prática;
- **Multiple Instruction Multiple Data (MIMD)**: múltiplas instruções provenientes de vários núcleos realizam operações em diferentes fluxos de dados. Este é o tipo de paralelismo mais utilizado atualmente, presente em virtualmente todos computadores. Máquinas deste paradigma podem ainda ser classificadas como computadores de memória compartilhada, computadores de memória distribuída ou híbridos.

2.1.1 Sistemas de Memória Compartilhada

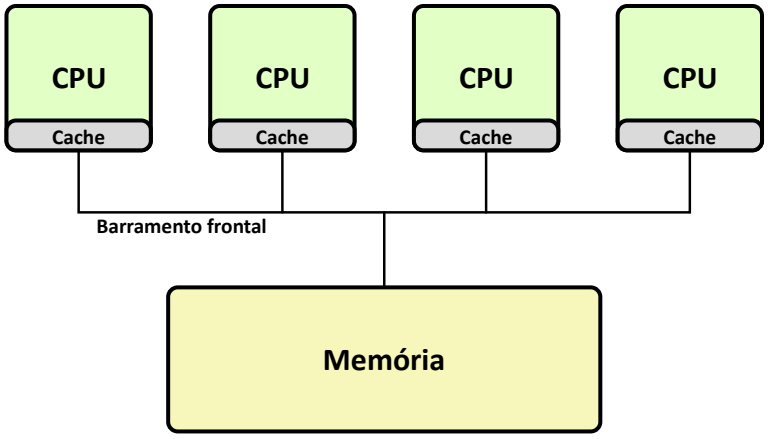
Segundo Foster (2019), em computadores de memória compartilhada, todos os processadores acessam um espaço de endereço físico de memória global. Assim, é possível a execução de diversos processos independentes que utilizam o mesmo recurso de memória, que pode ser acessado de maneira uniforme ou não-uniforme, dependendo da arquitetura adotada. A execução de múltiplos processos simultâneos conectados por um espaço de memória compartilhado é formalmente denominada de processamento paralelo ou Multiprocessamento Simétrico (*Symmetric Multiprocessing* – SMP) (Bertsekas; Tsitsiklis, 2015).

A Figura 5 apresenta um sistema de Acesso Uniforme de Memória (*Uniform Memory Access* – UMA) de SMP. Nesta configuração, cada núcleo possui um Barramento Frontal (*Frontside Bus* – FSB) individual responsável pela comunicação com a memória. A conexão entre um núcleo de processamento e a memória é roteada pelo *chipset* ou *northbridge* da Placa-Mãe (*Motherboard* – MOBO). O *chipset* tem como principal função

[2] AMD Alveo (2024) – https://www.amd.com/pt/products/accelerators/alveo.html
[3] NVIDIA Jetson (2024) – https://developer.nvidia.com/embedded/jetson-modules

impor a coerência entre os *caches*, uma memória de alta velocidade localizada próxima ao processador, que armazena dados frequentemente acessados (Hager; Wellein, 2019).

Figura 5 – Sistema de memória compartilhada de acesso uniforme (UMA)

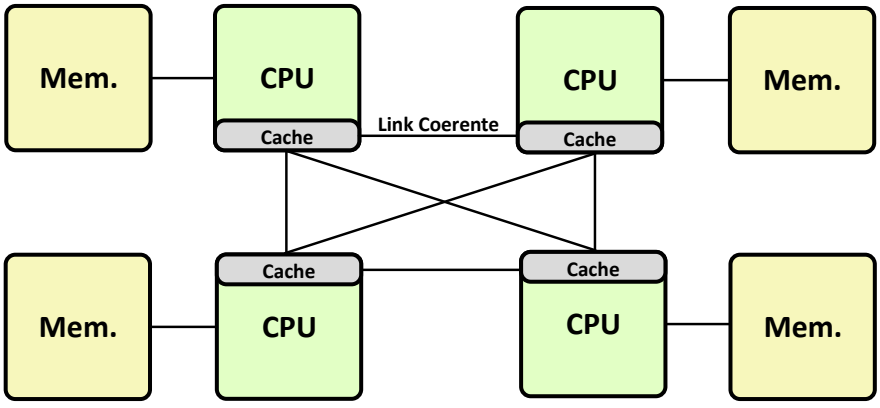


Fonte: Autor (2024).

A principal desvantagem do sistema UMA é que a escalabilidade está limitada pelo número de barramentos e largura de banda que podem ser acomodados pelo *chipset*. Por outro lado, o multiprocessamento simétrico permite acesso rápido e uma perspectiva simplificada do espaço de endereço de memória para programação de aplicações (Eijkhout, 2010).

No sistema de Acesso Não-Uniforme de Memória (*Non-Uniform Memory Access* – NUMA), exemplificado na Figura 6, cada processador tem acesso local a uma parte da memória do sistema, que pode ser acessada diretamente de maneira mais eficiente. O acesso à memória de outros processadores é realizado por meio de uma conexão coerente que mantém o *cache* de todos os processadores em sincronia (Hager; Wellein, 2019).

Figura 6 – Sistema de memória compartilhada de acesso não uniforme (NUMA)



Fonte: Autor (2024).

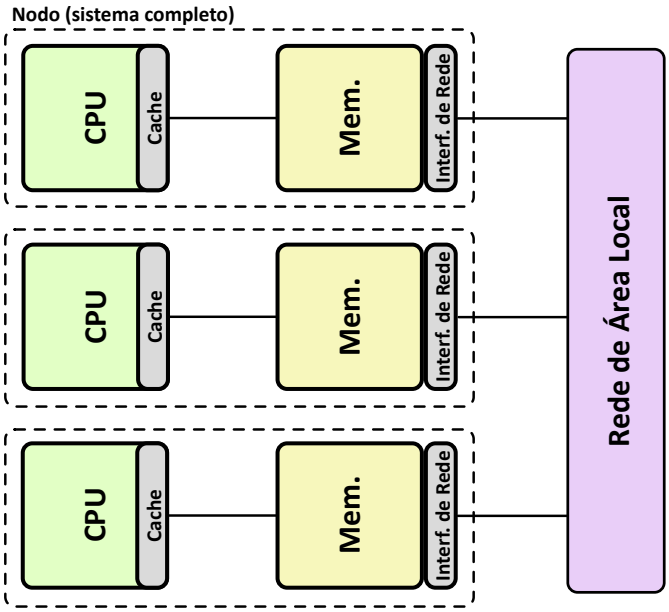
Sistemas NUMA são mais escaláveis em termos de largura de banda quando comparados com UMAs, mantendo ainda um espaço de endereço de memória único entre

os diferentes processadores. Entretanto, o fato de não haver uma conexão direta do FSB de cada processador com a memória de todo o sistema ocasiona um aumento significativo na latência de acesso e complexidade de desenvolvimento (Eijkhout, 2010).

2.1.2 Sistemas de Memória Distribuída

Os sistemas de memória distribuída consistem em diversas unidades de computação - denominadas de nós ou nodos - com processadores e espaços de memória únicos, conectados por uma interface de rede, como demonstra a Figura 7. Nesta configuração, cada nodo possui sistema operacional, sistema de arquivos e periféricos próprios, operando de maneira independente e sem acesso direto a memória de nenhum outro nodo do sistema (Trobec *et al.*, 2018).

Figura 7 – Sistema de memória distribuída



Fonte: Autor (2024).

Portanto, cada nodo do sistema executa tarefas de forma sequencial e se comunica com outros nodos por meio de envio e recebimento de mensagens. Aqui é possível traçar a distinção entre o processamento paralelo e distribuído:

- **Processamento Paralelo:** *refere-se à execução de múltiplos processos simultâneos conectados por um espaço de memória compartilhado em um mesmo sistema.*
- **Processamento Distribuído:** *caracteriza-se pelo agrupamento de múltiplos computadores em rede, permitindo a execução ubíqua de tarefas.*

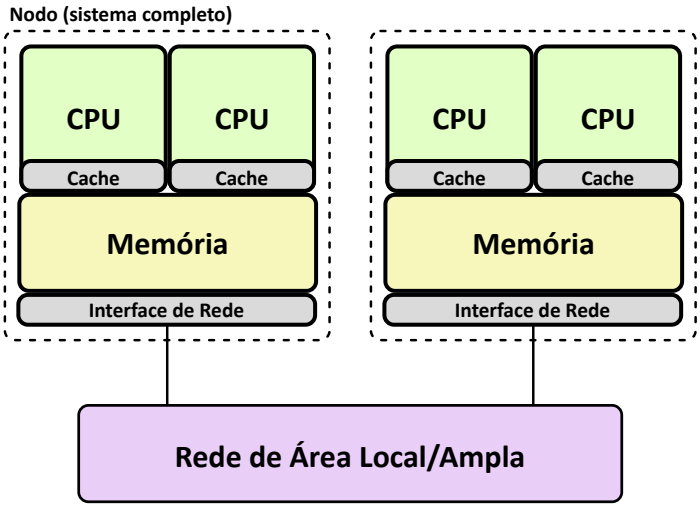
Conforme descrito por Hager e Wellein (2019), embora sistemas distribuídos sejam hoje raramente utilizados, eles marcaram o início dos HPCCs e permanecem relevan-

tes como modelo de programação para o padrão de passagem de mensagens, que será abordado nos capítulos seguintes deste trabalho.

2.1.3 Sistemas Híbridos

Supercomputadores modernos não são puramente de memória compartilhada ou distribuída, mas uma combinação de ambos. Na prática, supercomputadores são sistemas híbridos que usam como elementos básicos sistemas de memória compartilhada, agrupados por uma rede de alta velocidade, assim como sistemas distribuídos (Figura 8). Sistemas híbridos são facilmente escaláveis e possuem ótimo custo-benefício (Hager; Wellein, 2019).

Figura 8 – Sistema híbrido



Fonte: Autor (2024).

Entretanto, devido à sua natureza distribuída, os sistemas híbridos são ainda mais anisotrópicos do que os sistemas de memória compartilhada do tipo NUMA, pois o tempo de acesso à memória de um núcleo pode variar significativamente. Os dados podem estar armazenados na memória local do núcleo ou até mesmo na memória de um processador em um nodo de uma sub-rede distinta. Essa variação torna a topologia da rede em sistemas híbridos um aspecto crucial para garantir bom desempenho e escalabilidade.

2.2 REDES DE COMPUTADORES

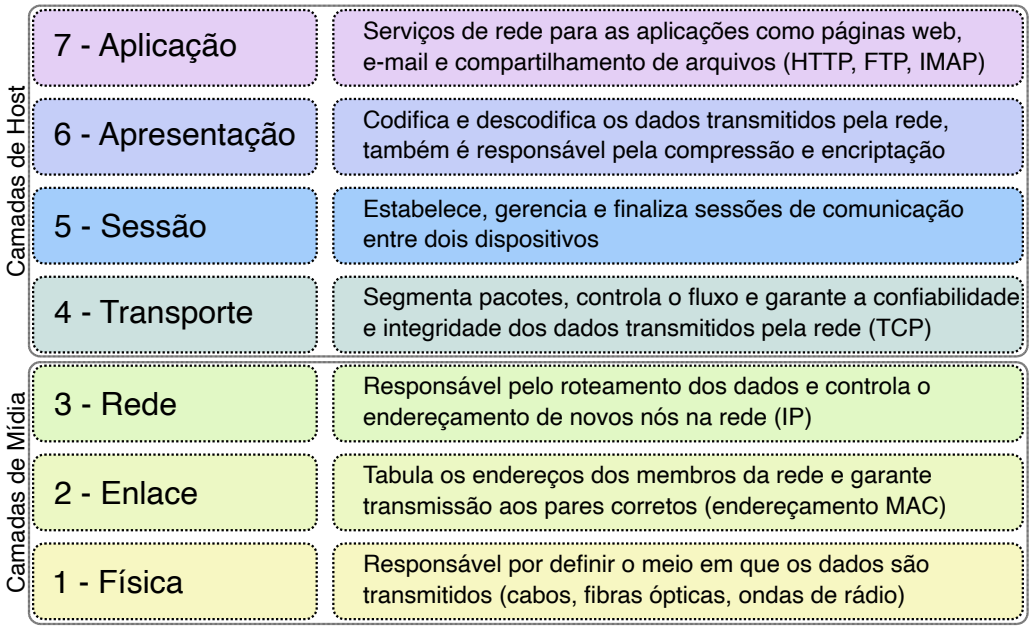
De maneira ampla, conforme descrito por Tanenbaum *et al.* (2021), uma Rede de Computadores pode ser descrita como uma conexão ou conjunto de conexões quaisquer de dois um mais computadores com o intuito de troca de dados. Computadores em uma mesma localidade geográfica - *e.g.*, residência, laboratório ou empresa - interligados por uma conexão, formam uma Rede de Área Local (*Local Area Network* – LAN). Quando

LANs em diferentes localidades geográficas são conectadas por um roteador ou ponte, formam uma Rede de Área Ampla (*Wide Area Network* – WAN), como a Internet.

Componentes comuns entre todas as redes são a presença de um meio físico para o transporte de dados (cabos, fibras ópticas, ondas de rádio); dispositivos de interconexão (hubs, comutadores, roteadores); um sistema de endereçamento de dispositivos, *i.e.*, uma forma de identificação unívoca dos integrantes da rede; e protocolos e *softwares* para codificação e controle do fluxo de dados transmitidos e recebidos (Sosinsky, 2009).

A Open Systems Interconnection (OSI), por meio da ISO/IEC 7498-1, determina que os componentes constituintes de uma rede podem ser abstraídos em sete camadas distintas, seguindo o modelo OSI, demonstrado na Figura 9. Essas camadas podem ser agrupas em camadas de mídia (física, enlace e rede), que tem como objetivo a transmissão física dos dados, bem como a identificação dos integrantes da rede e prevenção de colisões; e camadas de cliente ou *host* (transporte, sessão, apresentação e aplicação) focadas na preparação e formato de pacotes de dados, além de abstrair e gerenciar as sessões de conexão para o usuário final (Zimmermann, 1980).

Figura 9 – Camadas do modelo OSI de Redes



Fonte: Autor (2024).

Uma vez que as abstrações das camadas de *host* são construídas com base nas definições das camadas de mídia, compreender o melhor arranjo dos componentes físicos e suas interconexões se torna essencial. Essa disposição, conhecida como topologia, impacta diretamente o tipo de transmissão, a escolha de protocolos, a codificação e a apresentação de dados, além de influenciar o desempenho das aplicações conectadas à rede.

2.2.1 Topologias

As topologias descrevem como os dispositivos estão física e/ou logicamente distribuídos em uma rede de computadores. Esses dispositivos incluem nodos ou *endpoints*, como clientes e servidores, que enviam e recebem dados; dispositivos de interconexão, como roteadores e comutadores, que gerenciam o tráfego e garantem a segurança; e as conexões ou *links*, que são os cabos ou sinais pelos quais os dados são transmitidos. Em sistemas de alto desempenho, as topologias mais comuns são estrela, anel, malha, toro e hipercubo (Tanenbaum, 2009).

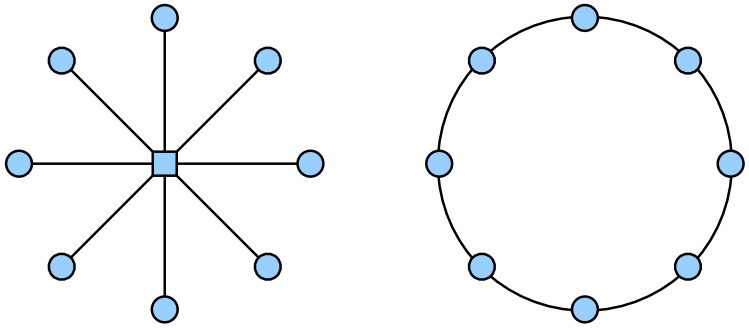
Na avaliação de projetos de topologias, os principais parâmetros de desempenho considerados são: (i) latência; (ii) largura de banda; (iii) diâmetro; e (iv) largura de banda de bisseção (Sarangi, 2021). Os termos largura de banda e latência são amplamente utilizados, mesmo por usuários leigos. Segundo Tanenbaum (2009), a largura de banda refere-se à taxa de dados transmitidos por unidade de tempo, enquanto a latência é o tempo total necessário para a transmissão de um pacote de tamanho predeterminado.

Ademais, ao considerarmos a menor distância - *i.e.*, o menor número de *links* para conectar dois nodos quaisquer - entre todos os pares de uma rede, a maior destas distâncias é definida como seu diâmetro. Em contrapartida, a largura de banda de bisseção é definida como a largura de banda agregada (soma da largura de banda de todos os *links*) das conexões cortadas para secionar uma rede em partes iguais. Sistemas distribuídos de alto desempenho destacam-se por sua assimetria e, os parâmetros de diâmetro e largura de banda de bisseção fornecem uma medida da anisotropia da rede em termos de latência e largura de banda (Jain *et al.*, 2016).

Na Topologia Estrela, todos os nodos estão conectados a um comutador central, por onde todo o tráfego de dados passa. Assim, o diâmetro da rede é constante e igual a dois, pois qualquer par de nodos está a duas conexões de distância. Para particionar a rede, basta remover o comutador central, tornando sua largura de banda de bisseção equivalente à largura de banda agregada do comutador. Segundo Sosinsky (2009), esta topologia é amplamente utilizada devido ao seu baixo custo e simplicidade de implementação. No entanto, sua escalabilidade é limitada pelo número de conexões que o comutador central pode suportar. Esse problema pode ser mitigado ao organizar topologias em forma de árvore, conectadas a outros comutadores.

Já na Topologia Anel, o comutador central é eliminado. Cada nodo se conecta a dois outros nodos, formando um círculo, possuindo uma conexão à direita e outra à esquerda. O protocolo *Token Ring* é usado para controlar a transmissão de dados e prevenir colisões. Um símbolo, ou *token*, é passado pelo anel e apenas o nodo detentor do símbolo tem permissão para transmitir os dados (Sosinsky, 2009). Neste caso, a maior distância é a de nodos diametralmente opostos, ou seja, há uma relação linear entre o número de nodos na rede e o diâmetro, tornando sua escalabilidade ruim em termos de latência. As topologias de estrela e anel são exemplificadas na Figura 10.

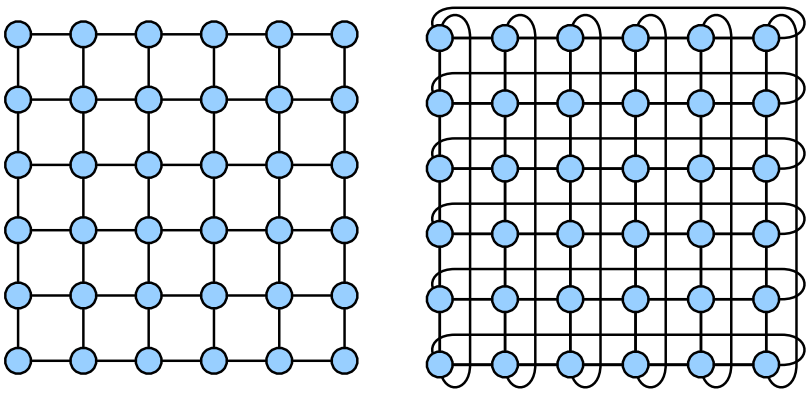
Figura 10 – Topologias Estrela e Anel



Fonte: Autor (2024).

De acordo com Tanenbaum *et al.* (2021), uma topologia altamente regular e escalável é a Topologia de Malha, também conhecida como Grade. Nela, o diâmetro da rede aumenta apenas com a raiz do número de nodos em contraste com a relação linear estabelecida pela topologia de anel. Alternativamente, a Topologia de Toro Duplo conecta as extremidades da malha fazendo com que o diâmetro da rede seja ainda menor e mais tolerante a falhas. Estas topologias apresentam um custo de implantação elevado quando comparadas com as topologias anteriores, especificamente devido ao *fan-out*[4], definido como o número de conexões entrando/saindo de um mesmo nodo. Ambas as topologias são apresentadas na Figura 11.

Figura 11 – Topologias Malha e Toro duplo



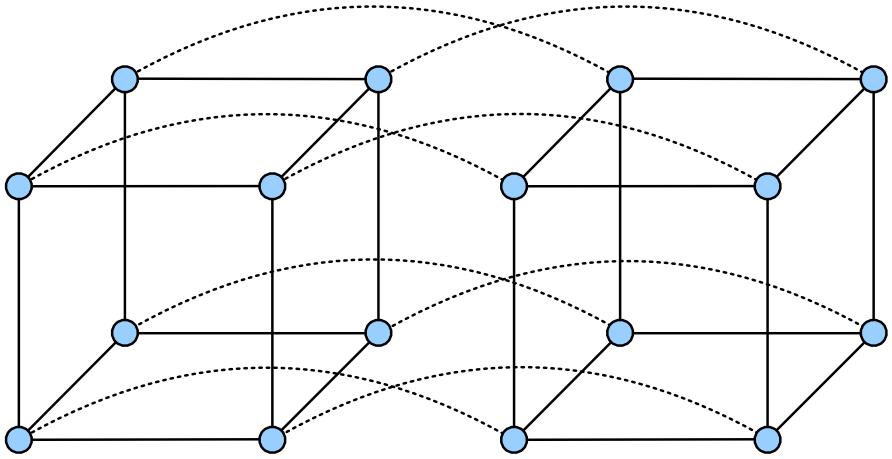
Fonte: Autor (2024).

Para a discussão da Topologia de Hipercubo, considere dois nodos ligados por um *link*. Duplicando esta rede e conectando os nodos correspondentes (mesmos vértices da estrutura duplicada) obtém-se um quadrilátero regular com arestas como *links* e vértices como nodos. Este processo pode ser iterado k vezes, formando assim um k-hipercubo. A Figura 12 mostra um hipercubo tetradimensional, formado por dois cubos ligados por seus nodos correspondentes. A topologia de hipercubo é especialmente interessante, pois seu

[4] Em última instância o *fan-out*, representa a quantidade de interfaces de rede e cabos necessários para conectar um *endpoint*.

diâmetro cresce proporcionalmente a $\log_2(N)$. Entretanto, a escalabilidade propiciada por esta topologia está associada a uma alta complexidade e custo de implementação, com um *fan-out* igual a dimensionalidade do hipercubo (Taubenblatt, 2011).

Figura 12 – Topologia de hipercubo tetradimensional



Fonte: Autor (2024).

A Tabela 2 apresenta uma comparação entre as diferentes topologias apresentadas. Contudo, é importante salientar que a escolha de uma topologia para sistemas HPC não deve se basear apenas nestes parâmetros, mas também nos equipamentos disponíveis, tamanho da rede, e tipo de carga de trabalho.

Tabela 2 – Características de diferentes topologias físicas de rede

Topologia	**# de links**	**Diâmetro**	**Bisseção**
Estrela	N	2	B
Anel	N	$N/2$	$2B$
Malha	$2N - 2\sqrt{N}$	$2\sqrt{N} - 2$	$2\sqrt{N}B$
Toro Duplo	$2N$	$\sqrt{N}$	$2\sqrt{N}B$
Hipercubo	$N\log_2(N)/2$	$\log_2(N)$	$NB/2$

N = número de nodos na rede
B = largura de banda de um *link*

Fonte: Autor (2024).

Para a implantação física dessas topologias de rede, diferentes tipos de tecnologias e equipamentos estão disponíveis, cada um adequado a requisitos específicos de desempenho e escalabilidade. O padrão Ethernet, amplamente utilizado em LANs, oferece uma solução de baixo custo e fácil implementação, sendo ideal para topologias estrela e anel em ambientes corporativos e residenciais. Para sistemas de alto desempenho, como em supercomputadores ou grandes *data centers*, de acordo com Taubenblatt (2011), tecnologias como InfiniBand[5] e OmniPath se destacam, proporcionando uma largura de banda

[5] NVIDIA InfiniBand (2024) – https://www.nvidia.com/pt-br/networking/products/infiniband/

significativamente maior e latências muito menores. Redes InfiniBand se destacam no contexto de HPC, pois são utilizadas em 60% das instalações no ranking TOP500 (2023).

2.2.2 Protocolos de Redes e Virtualização

Após estabelecida a topologia de rede, é fundamental garantir que a comunicação entre os dispositivos conectados ocorra de forma segura, eficiente e contínua. Isso exige uma pilha robusta de protocolos que atuam em diferentes camadas do modelo OSI. A identificação e o endereçamento unívoco dos dispositivos são gerenciados pelas camadas de enlace e rede, enquanto a transmissão confiável e coerente dos dados é assegurada pela camada de transporte.

O Controle de Acesso ao Meio (*Medium Access Control* – MAC) atua na camada de enlace, sendo responsável por controlar como os dispositivos em uma rede local acessam o meio físico e transmitem os dados. Cada dispositivo possui um endereço MAC único e independente da rede, utilizado para garantir que os pacotes de dados sejam entregues corretamente aos destinatários no nível de *hardware*. Além disso, o protocolo MAC gerencia o acesso ao meio compartilhado de forma a evitar colisões, especialmente em redes Ethernet, onde múltiplos dispositivos competem pelo uso do mesmo canal de comunicação (Black, 1993).

De acordo com Liang (2012), uma Rede de Área Local Virtual (*Virtual Local Area Network* – VLAN) complementa o protocolo MAC ao permitir a criação de topologias lógicas sobre uma infraestrutura física compartilhada. Esse conceito é crucial em ambientes de computação distribuída e corporativos, onde múltiplos sistemas e aplicações precisam compartilhar os mesmos recursos físicos sem interferir uns nos outros. A virtualização abstrai os componentes físicos da rede, permitindo a criação de VLANs que podem ser configuradas e gerenciadas de maneira flexível, escalável e independente.

O Protocolo da Internet (*Internet Protocol* – IP) opera na camada de rede e é responsável pelo roteamento dos pacotes de dados entre dispositivos. Cada dispositivo recebe um endereço IP, que serve como sua identificação lógica, permitindo a correta entrega dos pacotes ao destino final. Durante a transmissão, o IP também tem a função de fragmentar os pacotes para melhor adaptação às condições da rede, reagrupando-os no destino (Naugle, 1992). Embora o endereço MAC seja fixo e embutido no dispositivo de rede, em diferentes redes, este dispositivo pode ter, e provavelmente terá, diferentes endereços de IP.

Em conjunto com o IP, o Protocolo de Configuração Dinâmica de Host (*Dynamic Host Control Protocol* – DHCP) atua facilitando a atribuição de endereços de IP para dispositivos conectados. O DHCP distribui endereços de forma dinâmica e automática, garantindo que cada máquina receba um IP válido para comunicação (Black, 1993). O processo de atribuição é baseado no endereço MAC de cada dispositivo, permitindo que o servidor DHCP associe persistentemente um IP a um determinado nodo, garantindo que

o dispositivo possa se comunicar corretamente na rede sem intervenções. Esse protocolo simplifica o gerenciamento de grandes redes, reduzindo o risco de conflitos de endereçamento e facilitando o processo de provisionamento de novas máquinas (Naugle, 1992).

Na camada de transporte, o Protocolo de Controle de Transmissão (*Transmission Control Protocol* – TCP) garante que os dados sejam entregues de forma confiável e na ordem correta. O TCP estabelece uma conexão entre os dispositivos de origem e destino antes de iniciar a transmissão de dados. Além disso, implementa mecanismos de controle de fluxo, reconhecimento de pacotes e retransmissão de dados em caso de perda, assegurando que os pacotes cheguem ao destino sem erros e na sequência correta, tornando-o ideal para aplicações que exigem alta confiabilidade (Liang, 2012).

2.3 GERENCIAMENTO

Infraestruturas de processamento e rede formam as bases para a construção de um ambiente HPC. Entretanto, a implantação de um cluster também necessita de uma pilha adequada de *softwares* para abstração e gerenciamento destes recursos, como: um sistema de arquivos robusto que permita o compartilhamento de dados eficiente entre nodos e usuários; um escalonador para coordenar o uso de recursos computacionais e a execução de processos; aplicações como bibliotecas, simuladores e/ou serviços Web; e um orquestrador para o provisionamento desta pilha aos nodos e monitoramento dos recursos.

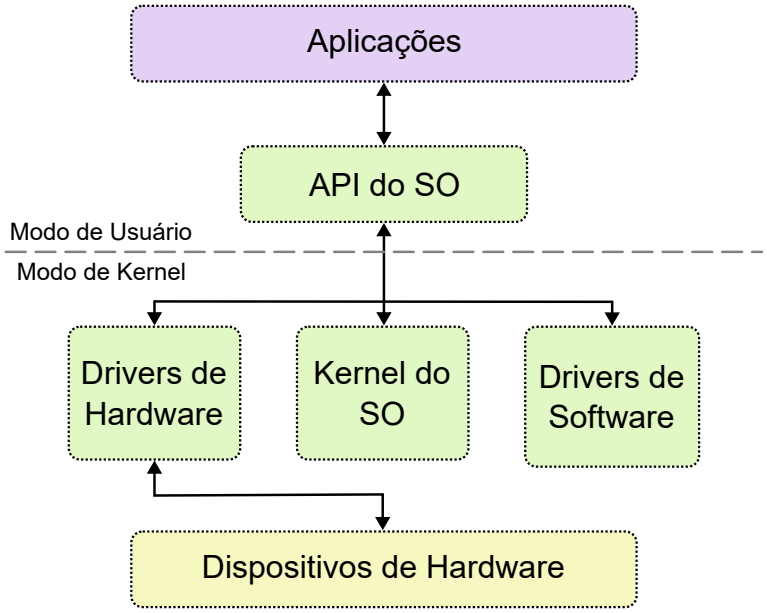
2.3.1 Sistema Operacional

Segundo Tanenbaum (2009), o Sistema Operacional (*Operating System* – OS) atua como a primeira camada de abstração, proporcionando uma interface para que programas - e programadores - se conectem aos dispositivos através de *drivers*. Além disso, o SO é fundamental para garantir a segurança na execução de processos, prevenindo acesso e modificações indevidas na memória através de espaços de endereços virtuais e dos modos de usuário e *kernel*.

A unidade central do OS é o *kernel*, que estabelece uma hierarquia entre os processos do sistema. As aplicações (*e.g.*, navegadores, editores de texto, interfaces gráficas) são executadas no modo de usuário. Cada processo em execução por essas aplicações possui endereços virtuais privados, o que significa que blocos específicos da memória do sistema só podem ser acessados ou modificados por esse processo. Em contrapartida, todo o código executado no modo de *kernel* possui acesso a um único espaço de endereço virtual de memória, e qualquer erro de acesso ou escrita por parte de um *driver* nesse nível pode comprometer o funcionamento do sistema como um todo (Silberschatz *et al.*, 2021).

Os drivers são componentes de *software* responsáveis pela conexão de dispositivos de I/O, como placas gráficas, interfaces de rede, discos rígidos e periféricos, ao kernel do sistema. Devido à sua natureza idiossincrática e dependente do fabricante, o SO fornece uma Interface de Programação de Aplicações (*Application Programming Interface* – API) que possibilita a interoperabilidade de dispositivos por meio de uma comunicação padronizada e eficiente (Silberschatz *et al.*, 2021). A Figura 13 ilustra como estes diferentes componentes operam para permitir a interação entre aplicações, *drivers* e dispositivos de *hardware*, assegurando a execução eficiente e segura de processos.

Figura 13 – Conexão entre diferentes componentes do Sistema Operacional



Fonte: Autor (2024).

Exemplos de sistemas operacionais comumente usados são GNU/Linux, FreeBSD, Windows e MacOS. De acordo com o ranking TOP500 (2023), todos os sistemas mais poderosos de HPC utilizam versões personalizadas/distribuições do Linux, devido à sua flexibilidade e suporte a ambientes distribuídos e paralelos. Por ser um sistema de código aberto, permite adaptação para atender às demandas de diferentes infraestruturas e cargas de trabalho. Entre as principais distribuições utilizadas em HPCCs estão CentOS[6], HPE Cray OS[7] e *Red Hat Entresprise Linux* (RHEL)[8].

2.3.2 Sistema de Arquivos

Assim como o OS coordena a integração entre dispositivos de *hardware* e *softwares*, o Sistema de Arquivos desempenha o papel de gerenciar como os dados são nomeados, armazenados, acessados e compartilhados, garantindo a interoperabilidade entre dis-

[6] CentOS (2024) – https://www.centos.org/
[7] HPE Cray OS (2024) – https://www.hpe.com/psnow/doc/a00067725enw
[8] RHEL (2024) – https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux/

cos rígidos, unidades de estado sólido ou outras mídias (Tanenbaum, 2009). Além disso, deve permitir interações de maneira simplificada e intuitiva com os dados armazenadas, fornecendo uma camada de abstração transparente ao usuário.

Sistemas de arquivos precisam oferecer características como desempenho, para permitir o rápido acesso e armazenamento de dados; segurança, para garantir que apenas usuários autorizados possam acessar informações; e confiabilidade, para assegurar que dados não sejam corrompidos ou perdidos durante a execução de operações. Entretanto, ambientes HPC e bases de dados massivas, estabeleceram novos desafios, como compartilhamento rápido e seguro, escalabilidade, gerenciamento de grandes conjuntos de dados e necessidade de tolerância a falhas em maior escala. Em HPCCs sistemas de arquivos compartilhados e/ou distribuídos são empregados como forma de superar esses desafios. Assim, de acordo com Hager e Wellein (2019) novos requisitos são particularmente críticos para esses ambientes, como:

- **Acesso concorrente**: Em clusters, múltiplos nodos precisam ler e gravar dados simultaneamente. Um sistema de arquivos deve suportar esse acesso massivo sem criar gargalos.
- **Escalabilidade**: À medida que o número de nodos no cluster cresce, o sistema de arquivos precisa ser capaz de escalar de forma eficiente, tanto em termos de armazenamento quanto em capacidade de processamento de I/O.
- **Tolerância a falhas**: Com muitos nodos trabalhando em conjunto, falhas de *hardware* são inevitáveis. O sistema de arquivos precisa ser capaz de continuar operando mesmo na presença dessas falhas, sem perda de dados.
- **Alta taxa de transferência**: Operações de I/O devem ser otimizadas para garantir que grandes volumes de dados possam ser lidos e gravados rapidamente, minimizando o impacto no desempenho geral do cluster.

Sistemas de arquivos distribuídos e compartilhados diferem na maneira como abordam esses desafios. Sistemas de arquivos compartilhados como *Network File System* (NFS) e *Server Message Block* (SMB) permitem o acesso, através de uma rede compartilhada, de diferentes usuários e/ou sistemas a um mesmo conjunto de dados, armazenado de maneira centralizada em um único local físico. Tanto no NFS quanto no SMB os arquivos compartilhados na rede são acessados de maneira transparente como se estivessem no armazenamento local. Apesar da simplicidade de implementação e manutenção, devido a sua natureza centralizada, apresentam um ponto único de falha e a medida que o número de nodos/clientes aumenta gargalos na taxa de transferência e colisões no acesso são inevitáveis (Thanh *et al.*, 2008).

Neste contexto, os sistemas de arquivos distribuídos apresentam uma alternativa mais eficaz no cumprimento dos requisitos listados ao dividir e espalhar os dados em diferentes nodos. Esse tipo de sistema melhora a escalabilidade, a redundância e o desempe-

nho, pois os dados podem ser acessados e processados em paralelo, em contraste com abordagem multiplexada de sistemas compartilhados. O *Hadoop File System* (HDFS) é amplamente usado em Clusters de Alta Disponibilidade (*High Availability Clusters* – HAs), pois automaticamente cria cópias de arquivos em múltiplos nodos para redundância, e por ser baseado no modelo *MapReduce*[9] é otimizado para operações que requerem grande taxa de transferência, como a execução de processos em lote (Shvachko *et al.*, 2010). Em contrapartida, o *Lustre File System* é voltado para supercomputadores científicos, pois permite a interconexão de dados durante processamento com baixa latência, favorecendo programas baseados em passagem de mensagens. Na realidade, HPCCs de grande porte, geralmente, possuem múltiplos sistemas de arquivos, cada um otimizado para uso em uma determinada carga de trabalho (Piernas *et al.*, 2007).

2.3.3 Provisionamento

Os maiores supercomputadores científicos, como Frontier, Fugaku e LUMI[10], possuem milhares de nodos de computação. Configurar cada um desses nodos manualmente seria extremamente trabalhoso, até mesmo para equipes compostas por dezenas de administradores de sistemas. Nesse contexto, orquestradores, ou *softwares* de provisionamento, desempenham um papel fundamental, automatizando o processo de configuração, gerenciamento e monitoramento de ambientes HPC. Instaladas no nodo mestre, essas ferramentas têm como objetivo otimizar a administração de grandes clusters, facilitando a implantação e manutenção de diversos nodos eficientemente (Schaffer *et al.*, 2009).

Ferramentas de provisionamento como Warewulf[11] e *Extreme Cloud Administration Toolkit* (xCAT)[12] fornecem um conjunto abrangente de funcionalidades. Conforme Kurtzer *et al.* (2012), a principal função dos orquestradores é a de geração de imagens de OS que incluem configurações personalizadas, bibliotecas, compiladores e pacotes de *software* específicos (*e.g.*, *drivers*, *softwares* de rede, sistemas de arquivos, escalonadores de tarefas). A inicialização dos nodos com as imagens personalizadas é realizada por meio do Ambiente de Pré-execução (*Preboot Execution Environment* – PXE), outro recurso em ambientes de provisionamento. O PXE permite que os nodos inicializem via rede, diretamente a partir do nodo mestre, sem a necessidade de mídias físicas como discos ou pendrives. Isso facilita a instalação remota e o gerenciamento de sistemas operacionais e acelera a implementação de novos nodos em grandes clusters (Rankin, 2008). Após inicializado o OS, também é possível a execução de um comando especifico no cluster inteiro ou subconjuntos específicos de nodos, minimizando a complexidade administrativa e o risco de erros.

---

[9] Ambos *MapReduce* e Passagem de Mensagens são modelos de programação paralela que serão discutidos no capítulo seguinte.

[10] Sistemas com maior desempenho no LINPACK na América, Ásia e Europa, respectivamente

[11] Warewulf (2024) – https://warewulf.org/

[12] xCAT (2024) – https://xcat.org/

Algumas placas-mãe possuem microcontroladores especializados denominados Baseboard Management Controller (BMC) que permitem o monitoramento remoto, independente da inicialização do sistema operacional. Orquestradores, por meio do BMC, permitem o gerenciamento direto do *hardware*, possibilitando operações como reinicialização, desligamento, ou diagnóstico de falhas em nodos sem que seja necessário o acesso direto ao sistema (Minyard, 2006).

2.3.4 Escalonador de Tarefas

Em ambientes HPC, as cargas de trabalho podem variar significativamente em complexidade, desde pequenos testes até simulações que demandam centenas de núcleos de processamento. O escalonador de tarefas é responsável por organizar, administrar e otimizar a utilização dos recursos disponíveis para a execução de tarefas (em inglês, *jobs*). O escalonamento em ambientes HPC é desafiador, já que esses sistemas geralmente atendem a múltiplos usuários com cargas de trabalho complexas, exigindo uma alocação inteligente e equilibrada de recursos (Skinner; Kramer, 2005).

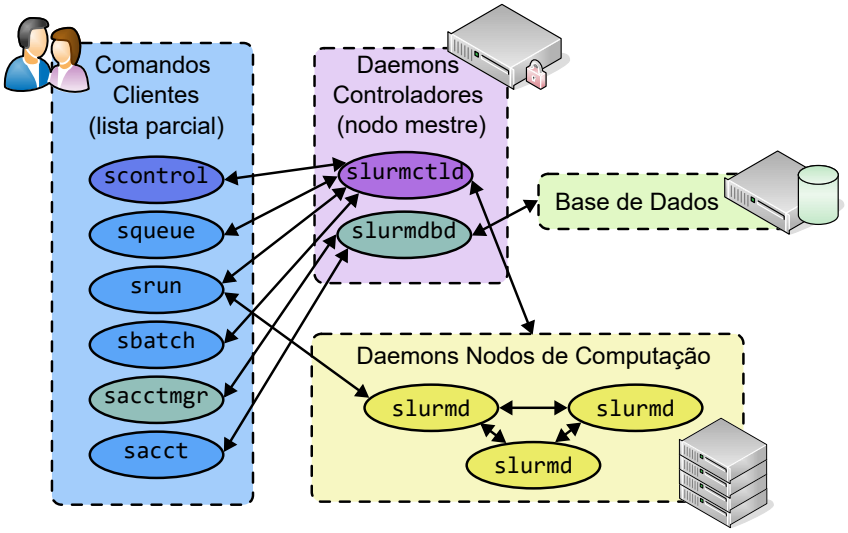
De maneira geral, um escalonador de tarefas para HPCCs deve: (i) criar políticas de usuários e grupos para estabelecer permissões e limites de uso dos recursos computacionais, definindo também prioridades na execução de tarefas; (ii) permitir que os usuários agendem e executem tarefas, especificando a quantidade de recursos desejados - *e.g.*, nodos, núcleos, GPUs - e o tempo de alocação, de acordo com os limites previamente estabelecidos; (iii) gerenciar filas de tarefas e realocar recursos ou ajustar a ordem de execução conforme necessário, balanceando cargas de trabalho e otimizando o desempenho geral do sistema; e (iv) possibilitar o monitoramento das tarefas em execução, além de registrar e armazenar estatísticas de tarefas finalizadas (Souza *et al.*, 2019).

O Simple Linux Utility for Resource Management (SLURM) é o escalonador de tarefas mais utilizado em supercomputadores, sendo responsável pela gestão de 65% dos sistemas no ranking TOP500 (2023). O SLURM opera através de *daemons*, processos que permanecem em segundo plano, permitindo que os usuários executem, agendem e monitorem *jobs* por meio de comandos simples, enviados ao *daemon* controlador (`slurmctld`) no nodo mestre. Esse nodo controlador se comunica e distribui as tarefas para os *daemons* nos nodos de computação (`slurmd`), conforme ilustrado na Figura 14. As configurações e estatísticas são armazenadas em um banco de dados relacional (*e.g.*, MySQL[13] e MariaDB[14]) e podem ser modificadas e consultadas pelos comandos `scontrol` e `sacctmgr` (Yoo *et al.*, 2003). Um exemplo adicional de escalonador é o OpenPBS, que também adota uma arquitetura baseada em *daemons* e oferece funcionalidades similares ao SLURM.

Os escalonadores de tarefas são componentes fundamentais para garantir o uso eficiente dos recursos, automatizando o processo de alocação e execução de *jobs* e ga-

[13] MySQL (2024) – https://www.mysql.com/
[14] MariaDB (2024) – https://mariadb.org/

Figura 14 – Componentes do Simple Linux Utility for Resource Management (SLURM)



Fonte: Adaptado de Yoo *et al.* (2003).

rantindo que múltiplos usuários e suas demandas sejam atendidas de forma ordenada. Em última instância, a camada com a qual o usuário final interage, seja uma pessoa ou uma aplicação, é o escalonador de tarefas, tonando-o uma peça central tanto na administração quanto utilização de clusters de alto desempenho. O Capitulo 5 trata exclusivamente do desenvolvimento de um *middleware* para integração do SLURM a processos de otimização baseados em populações.

2.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo, foram apresentados os fundamentos das arquiteturas de computação de alto desempenho, com ênfase nos conceitos centrais de processamento paralelo e distribuído, bem como nos elementos necessários para viabilizar sua implementação e operação. Inicialmente, definiu-se computação de alto desempenho (HPC) como o uso de sistemas computacionais compostos por múltiplas unidades de processamento cooperando para solucionar problemas complexos de forma eficiente, destacando sua relevância em áreas científicas e industriais.

Ao explorar as diferenças entre processamento paralelo e distribuído, discutiu-se que, no primeiro caso, múltiplos processos compartilham memória e recursos em uma única máquina, enquanto, no segundo, tarefas são distribuídas entre diferentes nodos conectados por uma rede, cada qual com sua memória local. Essa distinção enfatiza a importância das arquiteturas de memória compartilhada, distribuída e híbrida, cada uma oferecendo vantagens específicas dependendo do tipo de aplicação e da escalabilidade desejada.

Foi ressaltada a necessidade de uma infraestrutura de rede eficiente para permitir a comunicação entre os nodos de um sistema distribuído. Nesse contexto, foram aborda-

das as principais topologias de interconexão, como anel, malha, toro e hipercubo, além dos protocolos que suportam a troca de dados e a virtualização de recursos, permitindo o uso eficiente do ambiente computacional compartilhado.

Finalmente, foi discutida a importância de uma pilha de *softwares* que integre ferramentas para configuração, operação e gerenciamento do ambiente. Aspectos como o papel do sistema operacional, a escolha de sistemas de arquivos para armazenamento e acesso aos dados, as ferramentas de provisionamento para configurar e gerenciar nodos, e os escalonadores de tarefas para otimizar a execução de trabalhos foram abordados, evidenciando sua contribuição para a operação e administração de um sistema HPC.

3 DESENVOLVIMENTO PARA AMBIENTES HPC

O desenvolvimento de aplicações para ambientes HPC é fundamental em diversos campos da ciência e engenharia, pois permite a solução de problemas complexos e de grande escala, que seriam inviáveis em infraestruturas computacionais convencionais. Rauber e Rünger (2023) argumentam que o investimento significativo na construção e manutenção de sistemas HPC é justificado pela natureza paralelizável de muitos desses problemas, onde a execução simultânea de várias tarefas reduz o tempo de processamento e possibilita o armazenamento e o tratamento de grandes volumes de dados.

No entanto, a maximização dos recursos de um sistema HPC requer uma abordagem de desenvolvimento específica. A programação para ambientes HPC envolve práticas que vão além do uso de múltiplos núcleos de processamento. Desenvolver um programa para esses sistemas demanda estratégias para minimizar o tempo de comunicação entre nodos, otimizar o uso de memória compartilhada e distribuída, e gerenciar a sincronia/assincronia entre processos (Trobec *et al.*, 2018). Em muitos casos, o uso de processamento paralelo é, não apenas benéfico, mas mandatório, tanto pelo fato de que o uso de um único núcleo de processamento pode não ser capaz de realizar uma tarefa em um tempo razoável, quanto pela possibilidade de um único computador/sistema não possuir a quantidade de memória suficiente para solução de determinados problemas (Hager; Wellein, 2019).

Este capítulo explora as principais estratégias e modelos de programação que permitem desenvolver aplicações para sistemas HPC. Serão abordadas as técnicas de paralelismo de controle/funcional e de dados, que permitem adaptar e distribuir tarefas entre diferentes unidades de processamento. Em seguida, os padrões de Bifurcar e Juntar, Passagem de Mensagem e *MapReduce* - paradigmas mais empregados na programação paralela - serão apresentados. Por fim, serão discutidas métricas de desempenho que auxiliam na avaliação quantitativa e qualitativa de sistemas e programas paralelos, como o ganho de desempenho, a eficiência e a tolerância a falhas.

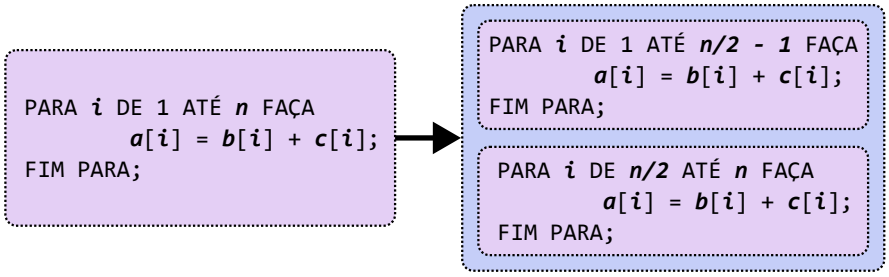
3.1 MODELOS DE PROGRAMAÇÃO PARALELA

Ao desenvolver ou adaptar programas para execução paralela, o primeiro passo, e por muitas vezes o mais crucial, é identificar quais tarefas podem ser de fato paralelizadas. De acordo com Pacheco (2011), em linhas gerais, os modelos de programação paralela podem ser classificados em duas abordagens principais: o paralelismo de dados e o paralelismo funcional ou de controle. Cada uma delas possui características próprias e é mais adequada para determinados tipos de problemas.

O paralelismo de dados envolve a discretização de um grande conjunto de dados em partes menores que podem ser processadas de maneira independente em diferentes

núcleos, processadores e/ou nodos (Chandra, 2001). A Figura 15 apresenta uma estrutura de laço que tem a simples funcionalidade de somar os valores de dois vetores em um terceiro vetor. Esta operação pode ser facilmente paralelizada ao realizar a operação de soma em diferentes segmentos do vetor simultaneamente.

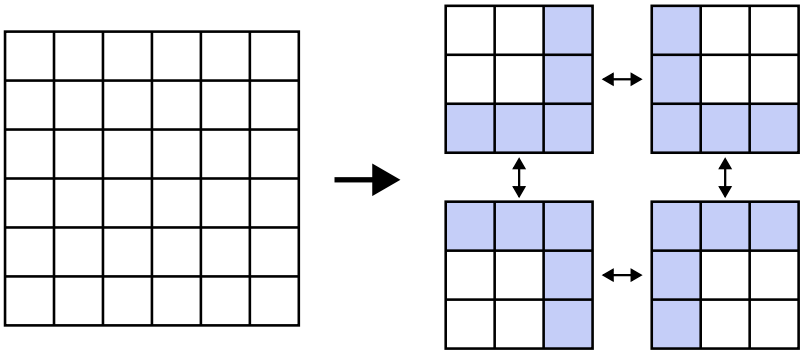
Figura 15 – Estrutura de laço com paralelização de dados



Fonte: Autor (2024).

Outro exemplo de paralelismo de dados é a decomposição do domínio de simulações físicas, como de escoamento de fluidos ou distribuição de tensões em sólidos. Neste caso, as equações que descrevem os fenômenos envolvidos são discretizadas e formam uma malha como demonstrado na Figura 16. Esta malha pode então ser decomposta em subdomínios que serão processados em paralelo (Bertsekas; Tsitsiklis, 2015). A decomposição deve ser realizada de maneira a gerar regiões com custos computacionais similares, balanceando a carga e evitando processos ociosos.

Figura 16 – Discretização de um domínio de simulação em subdomínios com interfaces



Fonte: Autor (2024).

Além disso, o paralelismo de dados requer o desenvolvimento de estratégias para minimizar o tempo de comunicação entre os processos, especialmente nas interfaces dos subdomínios. À medida que o processamento ocorre, os dados nas interfaces precisam ser trocados entre subdomínios adjacentes para garantir a consistência dos cálculos. Essa troca de informações entre interfaces pode gerar latência e afetar o desempenho da aplicação, especialmente em sistemas onde a comunicação entre nodos é mais lenta que o processamento local. Por isso, é necessário otimizar a frequência e o volume dessas comunicações (Rauber; Rünger, 2023).

Em contrapartida, no paralelismo funcional, a divisão do trabalho ocorre com base nas operações distintas que compõem o processamento. Ao invés de segmentar os dados, o programa é estruturado em etapas ou funções específicas que podem ser executadas de forma independente e atribuídas a diferentes nodos (Foster, 2019). O modelo mestre-trabalhador é um exemplo típico de paralelismo funcional. Neste modelo, um processo mestre distribui as tarefas entre múltiplos processos trabalhadores, que realizam o processamento e retornam os resultados ao mestre (Kasim *et al.*, 2008).

Contudo, devido ao fato de que cada processo executa uma tarefa distinta, o balanceamento de carga torna-se uma tarefa particularmente complexa. O modelo também depende de uma grande largura de banda à medida que o número de tralhadores aumenta, uma vez que toda comunicação é centralizada no mestre (Foster, 2019). Segundo Kasim *et al.* (2008), os padrões de programação paralela Bifurcar/Juntar, Passagem de Mensagens e *MapReduce* são comumente utilizados na implementação destes modelos de paralelização e apresentam características distintas quanto ao fluxo de dados e a distribuição de processos entre núcleos e nodos que serão apresentadas a seguir.

3.1.1 Bifurcar-Juntar/Multithreading

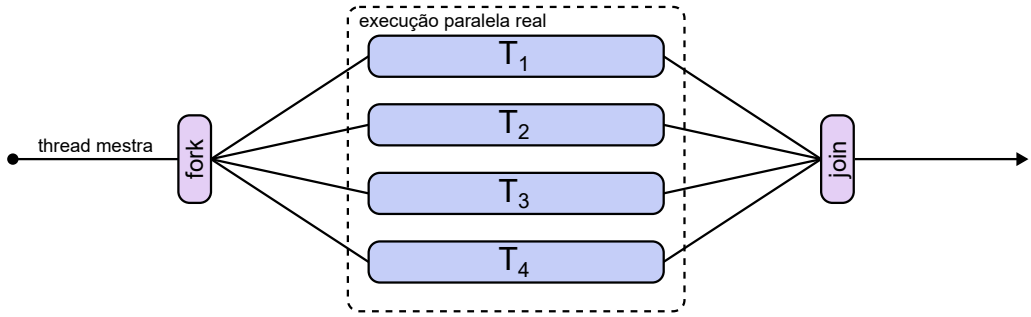
O modelo de programação Bifurcar-Juntar (ou *Fork-Join*, em inglês), é baseado no paradigma de dividir e conquistar, onde um problema é recursivamente dividido em subtarefas menores e independentes, chamadas *threads*, que são executadas simultaneamente. Em sistemas de memória compartilhada, as *threads*, diferentemente de processos, não possuem recursos de memória próprios, o que permite que dados, variáveis e estruturas estejam disponíveis para todas as *threads* sem a necessidade de cópias redundantes (Chandra, 2001).

Esse acesso direto é uma das principais vantagens do modelo Bifurcar-Juntar, pois reduz a sobrecarga de comunicação e facilita a sincronização entre *threads*. Entretanto, é necessário cuidado para evitar condições de corrida (em inglês, *race conditions*), *i.e.*, a violação de dados por operações de I/O a um mesmo endereço de memória concomitantes (Trobec *et al.*, 2018). De acordo com Chandra (2001), a execução paralela de múltiplas *threads* é também denominada de *multithreading*.

No contexto de processamento *multithreading*, é importante diferenciar o paralelismo real do paralelismo lógico. O paralelismo real, ilustrado na Figura 17, ocorre quando o *hardware* disponível permite a execução simultânea de várias *threads* em diferentes núcleos ou processadores. Já o paralelismo lógico se refere à divisão de uma tarefa em múltiplas *threads* que, apesar de programadas para executar simultaneamente, podem ser processadas de forma sequencial (Hager; Wellein, 2019).

Em sistemas com poucos núcleos, o paralelismo lógico permite que uma tarefa seja dividida em *threads*, mas nem todas serão executadas ao mesmo tempo. A operação de alternância entre elas é gerenciada pelo OS e é conhecida como troca de contexto (Fi-

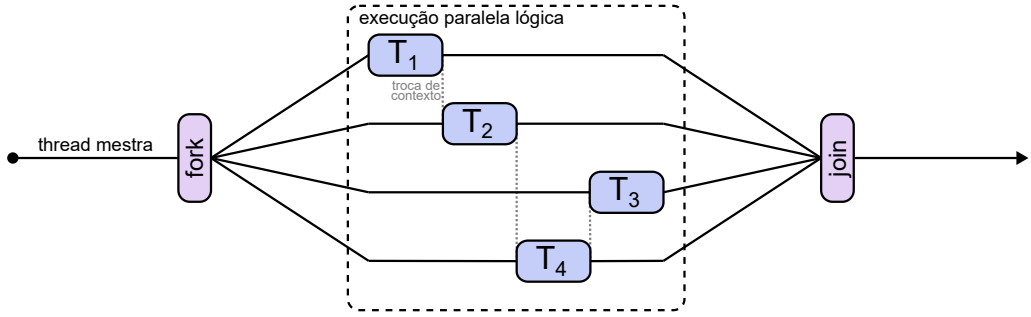
Figura 17 – Multithreading com paralelização real



Fonte: Autor (2024).

gura 18). O paralelismo lógico permite que o processador alterne rapidamente entre múltiplas *threads*, proporcionando a ilusão de paralelismo em sistemas com um número limitado de núcleos (Silberschatz *et al.*, 2021).

Figura 18 – Multithreading com paralelização lógica



Fonte: Autor (2024).

O *Open Multi-Processing* – (OpenMP)[1] é uma das principais ferramentas para implementar o padrão Bifurcar-Juntar, fornecendo uma API que permite criar e gerenciar *threads* de forma simplificada. A API possibilita a inserção de diretivas no código para indicar que determinadas seções ou laços devem ser paralelizados, dividindo automaticamente o trabalho entre as *threads* disponíveis (Chandra, 2001). No caso do exemplo de paralelização de laço da Figura 15, o OpenMP permite que as iterações sejam distribuídas entre os núcleos como demonstrado na Figura 19.

3.1.2 Passagem de Mensagens

A passagem de mensagens é uma técnica para a sincronização e a comunicação entre processos em sistemas distribuídos. Ela permite que processos, mesmo em ambientes onde não há espaço de memória compartilhado, sincronizem suas ações e troquem dados de forma eficaz, utilizando uma rede para transmitir informações. Essa abordagem é fundamental para a interoperação de processos em sistemas HPC, nos quais cada nodo

[1] OpenMP (2024) – https://www.openmp.org/

Figura 19 – Paralelização de um laço utilizando OpenMP

```
1  #include <stdio.h>
2  #include <omp.h>
3
4  void main(int argc, char **argv) {
5      //inicializa  o dos vetores
6
7      #pragma omp parallel for
8      for (i = 0; i < N; i++) {
9          a[i] = b[i] + c[i];
10     }
11 }
```

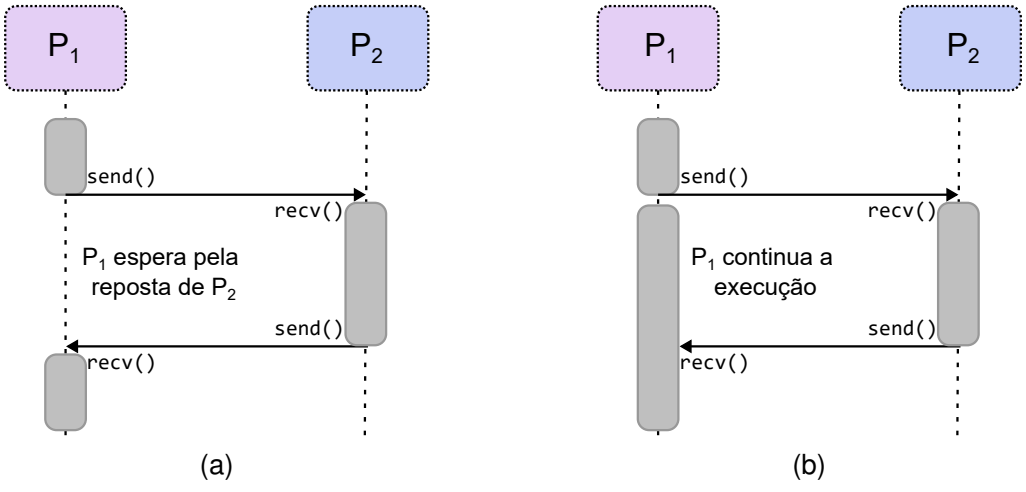
Fonte: Autor (2024).

do sistema pode funcionar de forma independente, mas precisa se comunicar para executar tarefas paralelas de forma coordenada (Trobec *et al.*, 2018).

O modelo de passagem de mensagens se baseia em duas funções básicas de envio e recebimento de dados, *i.e.*, `send()` e `recv()`. O processo de envio inicia a transmissão de uma mensagem a outro processo ou conjunto de processos, enquanto o processo de recepção capta a mensagem enviada, possibilitando que as informações/dados contidos nas mensagens sejam utilizadas ou passadas adiante entre processos (Gropp *et al.*, 1999).

Essa comunicação pode ocorrer tanto de forma síncrona quanto assíncrona como ilustrado na Figura 26. Na execução síncrona o processo emissor é suspenso até que haja uma confirmação ou retorno da mensagem enviado ao destinatário. Em contrapartida, na execução assíncrona o processo emissor continua sua execução logo após enviar a mensagem, sem esperar pela confirmação de recebimento (Pacheco, 2011).

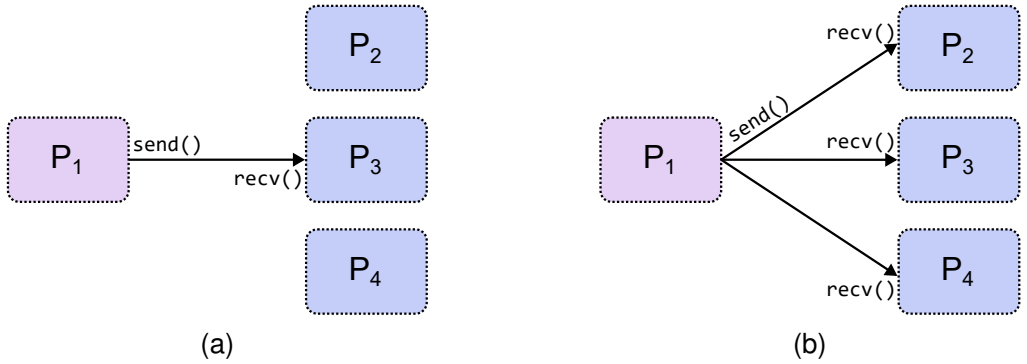
Figura 20 – Comunicação entre processos de forma (a) síncrona e (b) assíncrona



Fonte: Autor (2024).

Outro aspecto importante da passagem de mensagens é o tipo de comunicação, que pode ser *point-to-point* ou *multicast*/*broadcast* (Figura 21). Na comunicação *point-to-point*, uma mensagem é enviada diretamente de um processo a outro, facilitando a troca de dados específicos entre pares de processos. Já a comunicação *broadcast* permite que um processo envie uma mensagem a múltiplos processos simultaneamente, o que é útil em operações onde vários processos precisam receber os mesmos dados, como na troca de variáveis/instruções globais (Pacheco, 2011).

Figura 21 – Passagem de mensagens (a) point-to-point e (b) broadcast



Fonte: Autor (2024).

Para padronizar a paralelização e a comunicação de processos baseada na passagem de mensagens, a Interface de Passagem de Mensagens (*Message Passing Interface* – MPI) é amplamente utilizado. O MPI é o principal padrão adotado para a passagem de mensagens e fornece uma interface robusta e flexível, que permite desenvolver aplicações de alto desempenho e escaláveis. Uma das grandes vantagens do MPI é sua capacidade de automaticamente selecionar o meio de interconexão mais eficiente para a comunicação, que pode variar entre memória compartilhada/*threads* para processamento local, InfiniBand quando disponível, ou TCP/IP para passagem de mensagens em redes Ethernet (The MPI Forum, 1993).

Além disso, o MPI garante a entrega segura, contínua e ordenada das mensagens entre os processos. Isso significa que as mensagens enviadas entre dois processos seguirão a ordem correta de chegada, e os dados transmitidos estarão íntegros, independentemente da topologia de interconexão ou arquitetura dos nodos (Gropp *et al.*, 1999). Devido a essas garantias, o MPI tornou-se uma ferramenta ubíqua no desenvolvimento de aplicações distribuídas e paralelas em sistemas HPC, com duas implementações principais, o Open MPI[2] e o MPICH[3], que podem ser usados de maneira intercambiável.

[2] Open MPI (2024) – https://www.open-mpi.org/
[3] MPICH (2024) – https://www.mpich.org/

3.1.3 MapReduce

O modelo de programação *MapReduce* é amplamente utilizado para processamento de grandes volumes de dados, aplicando uma estrutura que se aproveita da proximidade dos dados nos sistemas de arquivos distribuídos. Esse modelo foi projetado para simplificar a divisão do trabalho em sistemas de múltiplos nodos, particularmente em cenários onde o volume de dados é tão grande que torna inviável o processamento em uma única máquina. Em sua essência, o *MapReduce* realiza operações de particionamento, mapeamento, embaralhamento e redução, transformando e condensando dados complexos em uma série de passos eficientes (Kasim *et al.*, 2008).

O processo se inicia com a fase de particionamento, onde o conjunto de dados é dividido em blocos menores, com o objetivo de maximizar a localidade e reduzir a latência de leitura, pois cada fragmento é processado localmente no nodo onde se encontra. Em seguida, ocorre a fase de mapeamento, na qual uma função é aplicada a cada fragmento dos dados, produzindo uma série de pares chave-valor que serão organizados conforme sua categoria ou significado. Esta fase de mapeamento tem o função de extrair e organizar as informações necessárias a partir de conjuntos de dados brutos, distribuindo a carga entre diversos nodos do sistema (Dean; Ghemawat, 2008).

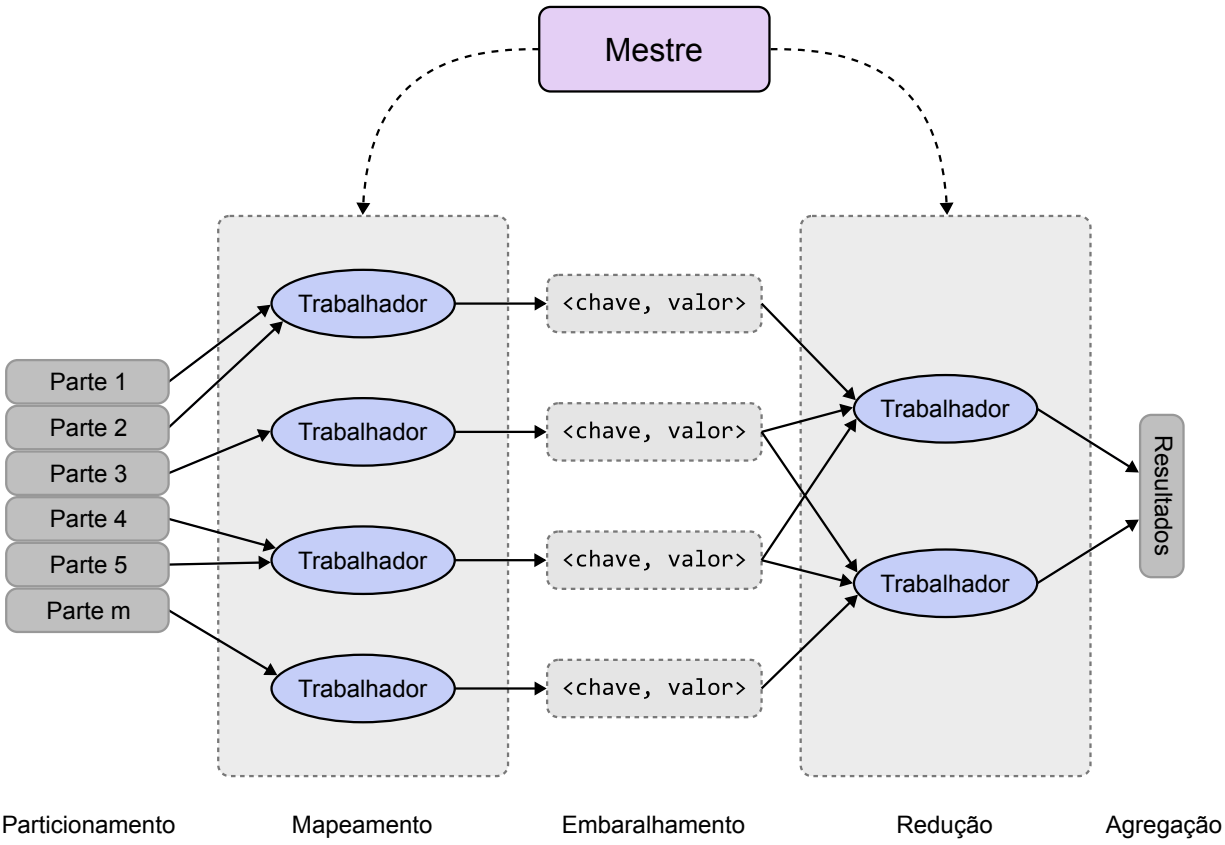
A fase seguinte é o embaralhamento, onde os pares chave-valor gerados são agrupados e reorganizados de acordo com suas chaves, permitindo que dados semelhantes sejam consolidados para a fase final. O embaralhamento é fundamental para garantir que cada nodo receba todos os dados relevantes para a etapa de redução, minimizando a troca de dados. Por fim, a etapa de redução aplica uma função de agregação sobre os dados embaralhados, consolidando-os em um conjunto de resultados finais que respondem ao problema inicial (Dean; Ghemawat, 2008).

O modelo *MapReduce*, com seu fluxo bem definido de operações e suporte a sistemas de armazenamento distribuído, é uma solução para processamento em larga escala, especialmente útil em áreas como análise de grandes bases de dados e na implementação de programas no modelo mestre-trabalhador (Rauber; Rünger, 2023). A Figura 22 apresenta uma visão geral do fluxo de operações do *MapReduce*, destacando as interdependências entre as fases e o fluxo de dados ao longo do processo.

3.2 MÉTRICAS DE DESEMPENHO

Avaliar o desempenho de sistemas e programas HPC garante que o desenvolvimento, a escolha de modelos de paralelismo e o uso de recursos seja otimizado e que a execução paralela entregue ganhos reais mensuráveis. Métricas de desempenho permitem mensurar a eficiência do programa em diferentes aspectos, verificando se os recursos computacionais disponíveis são aproveitados de maneira adequada, se o aumento de recursos resulta em melhorias proporcionais e se o programa é capaz de manter a robus-

Figura 22 – Fluxo de operações no padrão MapReduce



Fonte: Adaptado de Dean e Ghemawat (2008).

tez mesmo em casos de falhas em processos ou operações específicas (Hager; Wellein, 2019).

A seguir, são abordadas métricas que permitem uma visão abrangente de quão bem um sistema ou programa HPC realiza seu processamento, a saber: o desempenho máximo e eficiência, que mensuram o aproveitamento dos núcleos, processadores e nodos que compõem a infraestrutura; medidas de ganho de desempenho, que apresentam formas de avaliar o ganho de desempenho em comparação a uma implementação serial, o que permite entender o impacto real do paralelismo em termos de redução de tempo de execução; escalabilidade, que mede como o desempenho do programa cresce conforme mais recursos computacionais são disponibilizados; e tolerância a falhas, uma característica que garante que o sistema mantenha sua funcionalidade mesmo na presença de problemas ou interrupções em partes do processo.

3.2.1 Eficiência

O desempenho de sistemas computacionais e de programas científicos pode ser avaliado pela quantidade de operações de ponto flutuante realizadas por segundo, Flops/s

(*Floating Point Operations per Second*). Esse indicador é utilizado para quantificar a capacidade de processamento numérico de um sistema, especialmente no contexto de aplicações intensivas em operações matemáticas complexas como as encontradas em problemas de engenharia (Kindratenko; Trancoso, 2011).

O desempenho máximo teórico (R_{peak}) de um sistema corresponde ao maior valor possível de Flops/s que poderia ser atingido, considerando apenas a velocidade dos processadores e seu conjunto de instruções como demonstrado na Equação 1. Por exemplo, em processadores Intel Xeon da geração Haswell ou superiores, o conjunto de instruções *Fused Multiply-add Operation* – (FMA) permite 16 operações de ponto flutuante por ciclo de processamento, portanto, para um processados Intel Xeon E5-2630v3@2,4Ghz, com oito núcleos de processamentos, o desempenho máximo teórico calculado seria de $307{,}2$ GFlops/s, conmo mostra a Equação 2 (Dolbeau, 2018).

$$R_{peak} = \#\text{ processadores} \times \#\text{ núcleos/processador} \times \text{freq. } \textit{clock} \times \text{instruções/ciclo} \quad (1)$$

$$R_{peak,E5-2630} = 1 \times 8 \times 2{,}4 \cdot 10^9 \times 16 = 307{,}2 \cdot 10^9 \text{ Flops/s} \quad (2)$$

Entretanto, alcançar esse desempenho teórico é um desafio significativo devido a limitações práticas que envolvem operações de I/O, comunicação entre processos e o tempo necessário para acessar a memória. Tais fatores reduzem o desempenho efetivo e criam um desvio entre o desempenho máximo teórico e o desempenho obtido na prática (Dolbeau, 2018). Para mensurar essa discrepância, define-se a eficiência do sistema[4] (η_{sist}) como a razão entre o desempenho máximo real (R_{max}) alcançado por um programa e o desempenho máximo teórico (R_{peak}) do sistema (Equação 3). Uma eficiência alta indica que o sistema está utilizando de forma eficaz seus recursos, enquanto valores mais baixos podem sugerir gargalos, como comunicação excessiva ou limitações de memória, que impedem a plena utilização dos recursos de *hardware* (Kindratenko; Trancoso, 2011).

$$\eta_{sist} = \frac{R_{max}}{R_{peak}} \quad (3)$$

Diferentes *benchmarks* são utilizados para avaliar o desempenho e a eficiência de supercomputadores, destacando-se o *Basic Linear Algebra Subprograms* – (BLAS)[5] e o *High Performance LINPACK* – (HPL)[6] . Esses *benchmarks* aplicam operações de álgebra linear intensivas e permitem comparar o desempenho de diferentes sistemas com métricas padronizadas. A Tabela 3 destaca o desempenho e eficiência de alguns dos sistemas listados no ranking TOP500 (2023), com valores típicos de η_{sist} variando entre 45% e 85%, com enfase para o supercomputador Pégaso, sistema brasileiro melhor colocado.

[4] Neste contexto, eficiência do sistema não deve ser confundida com a eficiência do programa, também

Tabela 3 – Desempenho e Eficiência de Sistemas HPC no TOP500

Posição	Sistema	# núcleos	R_{max} [TFlops/s]	R_{peak} [TFlops/s]	η_{sist} [%]
1	Frontier	8.699.904	1.206,00	1.714,81	70,33
2	Aurora	9.264.128	1.012,00	1.980,01	51,11
3	Eagle	2.073.600	561,20	846,84	66,27
4	Fugaku	7.630.848	442,01	537,21	82,28
5	LUMI	2.752.704	379,70	531,51	71,44
50	SuperMUC-NG	305.856	19,48	26,87	72,49
56	Pégaso*	233.856	19,07	42,00	45,40
100	DGX SuperPOD	127.488	9,44	11,21	84,21
250	Altair	63.360	3,53	5,88	60,03
500	HSUper	41.832	2,13	3,21	66,36

*Melhor supercomputador brasileiro no ranking

Fonte: Adaptado de TOP500 (2023).

3.2.2 Escalabilidade

Para além do aproveitamento eficiente dos recursos disponíveis, métricas de ganho de desempenho (ou *speedup*) e escalabilidade oferecem uma análise de como sistemas e programas paralelos se beneficiam do aumento na quantidade de unidades de processamento. O ganho de desempenho (Equação 4), é definido como a razão entre o tempo de execução de uma tarefa em modo serial (T_1) e o tempo de execução em paralelo com p unidades de processamento (T_p), e quantifica o impacto do paralelismo sobre o tempo total de processamento (Hager; Wellein, 2019).

$$S_p = \frac{T_1}{T_p} \tag{4}$$

No entanto, o ganho de desempenho é limitado por fatores que afetam a execução paralela de qualquer carga de trabalho, como a presença de operações interdependentes que devem ser realizadas em uma sequência específica, o que reduz a possibilidade de execução simultânea. Essas limitações, definidas como concorrência, contenção e coerência, restringem o aumento linear de desempenho à medida que mais recursos são adicionados. Concorrência refere-se ao limite intrínseco de tarefas que podem ser executadas ao mesmo tempo, enquanto a contenção e a coerência se referem, respectivamente, ao conflito pelo uso dos mesmos recursos e à necessidade de sincronização dos dados entre os processos (Gunther, 2008).

conhecida como escalabilidade.

[5] BLAS (2024) – https://www.netlib.org/blas/

[6] HPL (2024) – https://www.netlib.org/benchmark/hpl/

Diferentes modelos paramétricos são empregados para quantificar estas limitações. Segundo Amdahl (1967), mesmo que uma tarefa seja dividida em p unidades de processamento, uma fração irredutível σ permanece sequencial e, portanto, apenas a porcão restante (*i.e.*, $(1-\sigma)T_1$) pode ser paralelizada. Nesse caso, o ganho de desempenho máximo é dado pela Equação 5, que ainda pode ser simplificada para a versão mais comumente encontrada da Lei de Amdahl (Equação 6).

$$S_p = \frac{T_1}{\sigma T_1 + \left(\frac{1-\sigma}{p}\right) T_1} \tag{5}$$

$$S_p = \frac{p}{1 + \sigma\,(p-1)} \tag{6}$$

Para valores de p tendendo ao infinito, a Equação 6 converge para um valor finito determinado por σ, sugerindo que, mesmo com recursos computacionais ilimitados, o ganho de desempenho é limitado pela fração sequencial da tarefa, como mostrado na Equação 7:

$$\lim_{p\to\infty} S_p = \sigma^{-1} \tag{7}$$

A Lei de Amdahl presume que a carga de trabalho possui tamanho constante. Gustafson (1988) sugere que o tamanho da carga deva aumentar proporcionalmente ao número de processadores p, resultando em uma retomada, teoricamente, da natureza linear do ganho de desempenho. O modelo de Gustafson (Equação 8) propõe que o ganho de desempenho aumente de forma mais favorável, expressando-o em função de uma fração sequencial, mas considerando a carga paralelizável como dominante.

$$S_p = \sigma + (1-\sigma)p \tag{8}$$

A Lei Universal da Escalabilidade (*Universal Scalability Law* – USL), demonstrada na Equação 9, altera a Lei de Amdahl a partir de um termo de correção κ, que representa a sobrecarga causada pela imposição de coerência entre processos (Gunther, 1993). O termo introduz uma sobrecarga crescente quando muitos processadores são adicionados, fazendo com que, diferentemente das Equações 6 e 8 haja um máximo em S_p para um valor ótimo de processos paralelos p^* (Equação 10).

$$S_p = \frac{p}{1 + \sigma\,(p-1) + \kappa p(p-1)} \tag{9}$$

$$p^* = \sqrt{\frac{1-\sigma}{\kappa}} \tag{10}$$

O balanço entre o ganho de desempenho e os recursos utilizados é mensurado pela escalabilidade, também conhecida como eficiência paralela do programa. A escalabilidade, como mostra a Equação 11, é representada pela razão entre o ganho de desempenho (S_p) e o numero de processos paralelos executados (p). Alta escalabilidade indica que a maior parte do poder de processamento adicional contribui diretamente para a redução do tempo de execução, enquanto baixa escalabilidade sugere perdas significativas devido a fatores como sincronização, comunicação entre processos e desequilíbrio de carga (Hager; Wellein, 2019).

$$\varepsilon = \frac{1}{p}\frac{T_1}{T_p} = \frac{S_p}{p} \tag{11}$$

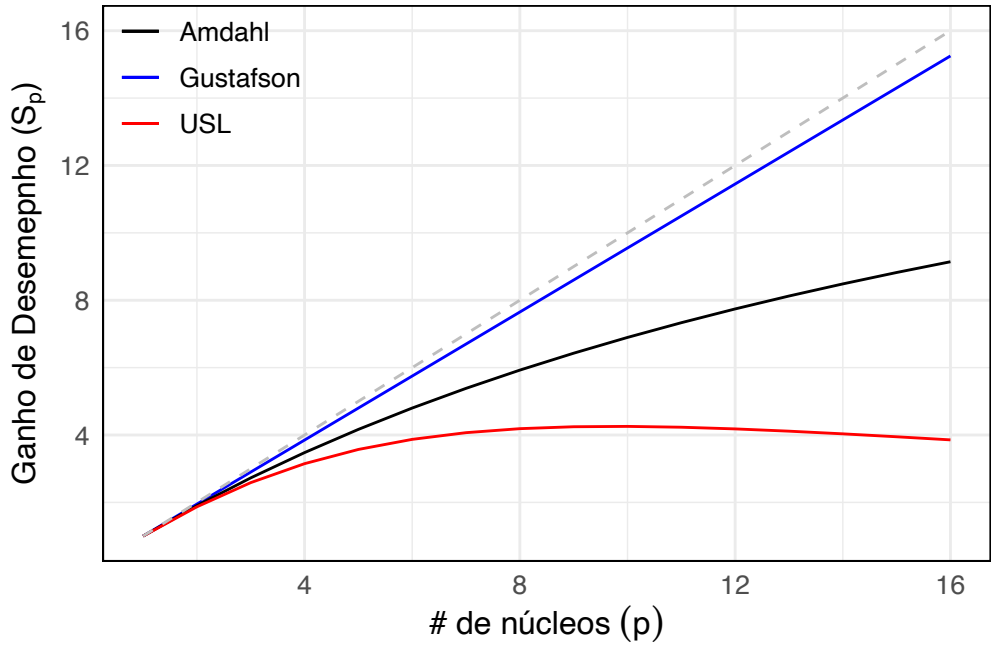
As Figuras 23 e 24 apresentam um comparativo no ganho de desempenho e escalabilidade modelados pela Lei de Amdahl, modelo de Gustafson e USL para valores de $\sigma = 0.05$ e $\kappa = 0.01$. É possível notar o impacto causado pelo coerência entre processos ao se comparar as curvas geradas pela Lei de Amdahl e USL até mesmo para pequenos valores de κ. Embora o modelo de Gustafson se aproxime do limite ideal (*i.e.*, $S_p = p$) desenvolver programas paralelos que apresentem ganho de desempenho linear tem se mostrado um desafio para os programadores destas aplicações (Gunther, 2008).

3.2.3 Tolerância a Falhas

Outro aspecto qualitativo na avaliação do desempenho de aplicações paralelas é a tolerância a falhas, que se refere à capacidade de um sistema de continuar funcionando adequadamente mesmo na presença de falhas em componentes de *hardware* ou *software*. Em sistemas de alto desempenho, a tolerância a falhas é essencial para garantir que operações de longa duração ou com requisitos críticos não sejam interrompidas, minimizando impactos negativos na execução de programas e no uso dos recursos computacionais (Jalote, 1994).

Em ambientes HPC, as falhas podem ocorrer em diferentes níveis, como nos nodos de processamento, na memória ou na comunicação entre processos. Um exemplo típico envolve a falha em um nodo durante uma execução distribuída: se um nodo deixa de responder ou apresenta erros, processos que dependem das informações calculadas por esse nodo podem ser interrompidos, levando a uma parada do sistema ou à necessidade de reiniciar a tarefa. Outro exemplo comum é a falha em redes de comunicação, que pode

Figura 23 – Speedup de diferentes modelos paramétricos ($\sigma = 0.05$, $\kappa = 0.01$)



Fonte: Autor (2024).

resultar na perda de pacotes ou em desconexões temporárias, afetando o sincronismo entre os processos paralelos (Jalote, 1994).

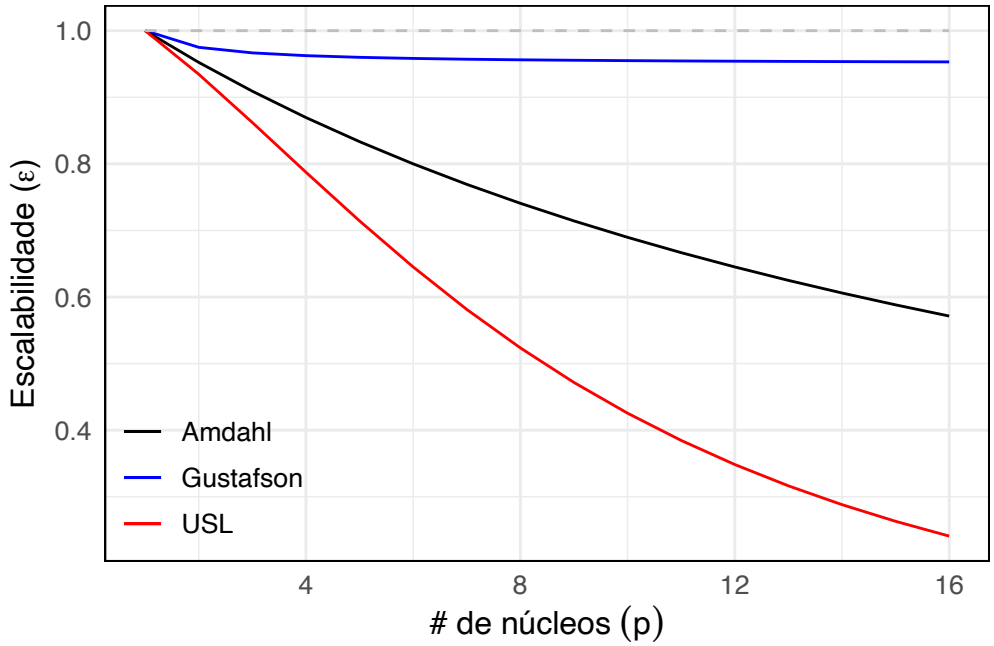
De acordo com Ledmi *et al.* (2018), para lidar com essas situações, diversas estratégias de tolerância a falhas podem ser implementadas, de acordo com a natureza do sistema e as características da aplicação. Uma técnica amplamente utilizada é o *checkpoint/restart*, na qual o estado do programa é periodicamente salvo em um armazenamento seguro, permitindo que, em caso de falha, o processo seja retomado a partir do último ponto de salvamento, em vez de reiniciar completamente a tarefa. Esse método reduz significativamente o tempo perdido e o custo de processamento em caso de interrupções.

Outra abordagem relevante é a replicação de dados e processos, em que múltiplas cópias de partes críticas do programa são executadas ou armazenadas em diferentes nós. Dessa forma, se uma réplica falhar, outra pode assumir sua função, garantindo a continuidade da execução sem grandes perdas. Além disso, o uso de algoritmos de detecção de falhas permite identificar rapidamente nodos ou processos problemáticos e realocar suas tarefas a outros componentes do sistema (Ledmi *et al.*, 2018).

3.3 CONSIDERAÇÕES DO CAPÍTULO

Foram abordados, neste capítulo, os principais aspectos do desenvolvimento para sistemas HPC, destacando os elementos fundamentais para a construção de aplicações eficientes e escaláveis. Inicialmente, foram apresentados os modelos de programação paralela, que oferecem diferentes abordagens para explorar o paralelismo inerente a sis-

Figura 24 – Escalabilidade de diferentes modelos paramétricos ($\sigma = 0.05$, $\kappa = 0.01$)



Fonte: Autor (2024).

temas HPC. O padrão bifurcar-juntar destacou-se por sua ênfase no compartilhamento de memória e no uso de *threads*, enquanto a passagem de mensagens foi abordada como a escolha predominante em arquiteturas de memória distribuída, com ferramentas como MPI amplamente utilizadas. Por fim, o modelo MapReduce evidenciou-se como uma abordagem eficaz em conjunto com sistemas de arquivos distribuídos para o processamento de grandes volumes de dados.

Além disso, o capítulo enfatizou a importância das métricas de desempenho na avaliação e otimização de aplicações para ambientes HPC. A eficiência foi explorada como um indicador da utilização proporcional dos recursos computacionais, enquanto a escalabilidade revelou-se útil para medir a capacidade de expansão do sistema à medida que mais recursos são adicionados. Foram analisados modelos paramétricos, que ajudam a prever o comportamento de sistemas conforme a carga ou o número de processadores aumenta. Já a tolerância a falhas, foi discutida com foco em estratégias para garantir a continuidade das operações diante de falhas inevitáveis, ressaltando a necessidade de técnicas como *checkpoints*, reinícios automáticos e redundâncias.

4 IMPLANTAÇÃO DO CLUSTER AQUA

O Núcleo de Simulação e Otimização de Sistemas Dinâmicos (NSO) dedica-se a pesquisa e desenvolvimento de soluções voltadas à simulação e otimização de sistemas complexos de engenharia, frequentemente associados a desafios computacionais intensivos. Alguns dos projetos em curso demandam recursos computacionais que ultrapassam a capacidade de estações de trabalho tradicionais (*i.e.*, *workstations*). Essa limitação, além de impactar os prazos de execução, dificulta/impede a exploração de modelos mais detalhados e robustos.

Com o objetivo de acelerar o processamento de simulações, viabilizar modelos computacionais mais avançados e oferecer um ambiente adequado ao desenvolvimento e experimentação científica, propôs-se a implantação de um *Commodity Cluster*. Essa abordagem aproveita a infraestrutura já existente no laboratório, composta por estações de trabalho heterogêneas, reduzindo custos enquanto proporciona um aumento significativo na capacidade de processamento.

Neste capítulo, a arquitetura da solução proposta – o Cluster Aqua – é descrita, abrangendo os componentes de *hardware*, as configurações de *software* e os aspectos de conectividade e segurança. Além disso, são apresentados os *benchmarks* realizados, demonstrando os resultados de desempenho obtidos e validando a infraestruturada HPC implantada.

4.1 HARDWARE

A seguir, são descritos os componentes de *hardware* utilizados na implantação do Cluster Aqua, incluindo os nodos de processamento, os dispositivos de interconexão e a topologia de rede. Uma visão geral da infraestrutura física é apresentada na Figura 25. Inicialmente, o cluster foi configurado com quatro nodos, todos aproveitados da infraestrutura existente no laboratório. Posteriormente, o sistema foi expandido para um total de oito nodos, visando uma expansão na capacidade de processamento. Contudo, um dos nodos (`nodo04`) encontra-se atualmente inoperante devido à queima de componentes causada por um pico de tensão durante o período de execução deste trabalho.

4.1.1 Nodos

O Cluster Aqua é composto por oito nodos, com funções e configurações distintas. O nodo mestre, responsável pela coordenação das tarefas e gerenciamento do cluster, é uma máquina HP Compaq 8200 Elite SFF (Figura 26a). Os sete nodos restantes desempenham funções de computação, sendo seis DELL Precision T7810 (dedicados a tarefas

Figura 25 – Visão geral do Cluster Aqua

Fonte: Autor (2024).

baseadas em CPUs, Figura 26b) e um DELL Precision T7910 (para tarefas com uso de GPUs, Figura 26c).

A heterogeneidade das configurações de *hardware* reflete a origem dos equipamentos. Três dos nodos de computação (`nodo01`, `nodo02` e `nodo03`) já integravam a infraestrutura do laboratório e possuem especificações idênticas, com maior capacidade de memória RAM em comparação aos demais. Os nodos adicionais, adquiridos posteriormente, ainda estão em processo de melhorias e atualização de componentes. Um exemplo é o `nodo05`, que possui capacidade para acomodar até quatro GPUs e já está equipado com quatro placas NVIDIA Quadro K620, destacando-se dos outros nodos por essa característica. As especificações completas dos nodos estão detalhadas na Tabela 4.

Tabela 4 – Configurações de hardware dos nodos do Cluster Aqua

Nodo	CPU	Núcleos	GPU	Memória
`mestre`	1×Core i5-2500	4 (1 : 4)	Radeon HD 6350	7GB
`nodo01`	2×Xeon E5-2630v3	32 (2 : 16)	Quadro K620	256GB
`nodo02`	2×Xeon E5-2630v3	32 (2 : 16)	Quadro K620	256GB
`nodo03`	2×Xeon E5-2630v3	32 (2 : 16)	–	256GB
`nodo04`	2×Xeon E5-2670v3	48 (2 : 24)	Quadro K620	256GB
`nodo05`	2×Xeon E5-2609v3	12 (2 : 6)	4×Quadro K620	128GB
`nodo06`	1×Xeon E5-2630v3	16 (1 : 16)	GeForce 9500GT	128GB
`nodo07`	1×Xeon E5-1607v4	4 (1 : 4)	Quadro K2000	32GB
Aqua	–	180	–	1.319GB

Fonte: Autor (2024).

Figura 26 – Nodos do Cluster Aqua: (a) nodo mestre, (b) e (c) nodos de computação

(a) HP Compaq 8200 Elite SFF (b) DELL Precision T7810 (c) DELL Precision T7910

Fonte: HP (2024) e DELL (2024).

4.1.2 Rede

A arquitetura de rede do Cluster Aqua foi projetada com base na topologia estrela, uma escolha fundamentada pela facilidade de implementação e pelo baixo custo associado. Apesar de suas limitações em termos de escalabilidade, essa configuração é adequada ao tamanho reduzido do cluster e à utilização do padrão Ethernet, que oferece desempenho suficiente para as demandas computacionais esperadas.

A conectividade externa do cluster é limitada ao nodo mestre, que está conectado à Internet por meio da RedeUFSC. Para suportar a comunicação interna entre os nodos, foi configurada uma VLAN dedicada, separada da rede externa. Essa abordagem exigiu a instalação de uma placa de rede adicional TP-Link TG-3468 (Figura 27) no nodo mestre, viabilizando a conexão do nodo a ambas as redes.

Os nodos do cluster estão interligados por um comutador gerenciável TP-Link TL-SG2210P (Figura 28), equipado com oito portas, cada uma com largura de banda de 1Gbit. A escolha por cabos Ethernet CAT6 garantiu a redução de interferências e o suporte a altas taxas de transferência, garantindo a comunicação confiável entre os nodos. Essa configuração assegura uma infraestrutura de rede eficiente e compatível com os dispositivos disponíveis no NSO.

Figura 27 – Placa de Rede adicional do nodo mestre

Fonte: TP-Link (2024a).

Figura 28 – Comutador gerenciável do Cluster Aqua

Fonte: TP-Link (2024b).

4.2 SOFTWARE

A configuração de *software* do Cluster Aqua foi amplamente baseada nas recomendações e receitas disponibilizadas pela comunidade *Open High Performance Computing* – (OpenHPC). O OpenHPC é um projeto de código aberto que fornece um conjunto abrangente de ferramentas, bibliotecas e pacotes pré-configurados para ambientes de computação de alto desempenho. Esse repositório consolidado simplifica a implantação e o gerenciamento de clusters ao disponibilizar soluções compatíveis com as melhores práticas (OpenHPC, 2018). Nesta seção as configurações adotadas são especificadas e justificadas. Os arquivos de configuração usados podem ser encontrados no Apêndice A deste trabalho.

4.2.1 Sistemas Operacional e de Arquivos

O repositório OpenHPC, em sua versão 2.3, oferece suporte a três distribuições Linux: CentOS 8, OpenSUSE Leap 15 e Rocky Linux 8. Após avaliar as opções, optou-se pela utilização do CentOS 8 devido a sua robustez e estabilidade. Essa escolha foi motivada pelo fato de o CentOS ser bem consolidado na comunidade HPC, com compatibilidade comprovada com as principais ferramentas e bibliotecas usadas nesse tipo de infraestrutura. Além disso, a integração com o repositório *Extra Packages for Enterprise Linux* – (EPEL) foi um fator determinante, pois esse repositório disponibiliza pacotes adicionais, como *softwares* de monitoramento e ferramentas administrativas, facilitando a instalação e configuração do ambiente do cluster.

Para o sistema de arquivos, optou-se pela utilização do NFS, uma solução simples e eficiente para compartilhar dados entre os nodos do cluster. O NFS foi configurado no nodo mestre, que hospeda tanto os diretórios pessoais dos usuários quanto os diretórios contendo ferramentas e aplicações utilizadas nas simulações e monitoramento dos nodos, com capacidade de armazenamento compartilhado de 1TB. Os nodos de computação não possuem dispositivos físicos de armazenamento (*e.g.*, discos rígidos) e utilizam apenas os diretórios compartilhados pelo NFS como sistema de arquivos.

Os diretórios de usuários são compartilhados com permissões específicas, permitindo leitura e escrita por parte dos nodos de computação. Em contrapartida, foram criados diretórios compartilhados exclusivamente para leitura, onde estão armazenados arquivos executáveis de aplicações específicas, como OpenFOAM, ANSYS, compiladores e ferramentas administrativas. Essa abordagem garante que os executáveis estejam centralizados e disponíveis para todos os nodos sem risco de modificações indevidas. Além disso, facilita a manutenção e o controle de versões das ferramentas utilizadas no cluster.

A montagem dos sistemas de arquivos nos nodos de computação é realizada automaticamente por meio do arquivo `/etc/fstab`, um arquivo de configuração padrão em sistemas Linux que contém informações sobre os sistemas de arquivos. As configurações

do arquivo restringem a criação de dispositivos especiais e a execução de binários com privilégios de superusuário, garantindo a segurança do ambiente.

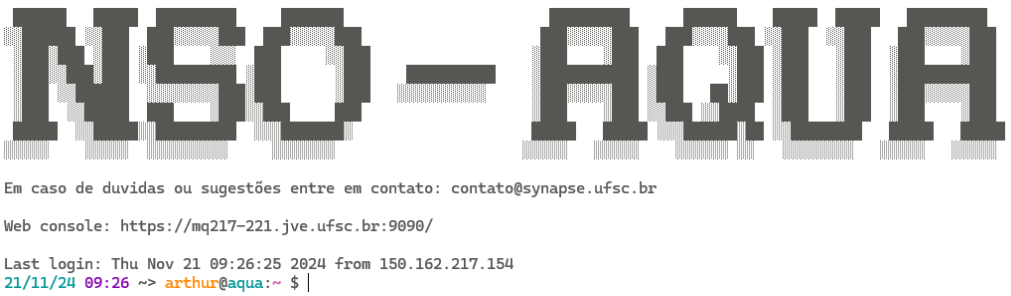
4.2.2 Conectividade e Segurança

A comunicação entre os nodos do cluster ocorre inteiramente por meio de uma VLAN interna, garantindo isolamento em relação à RedeUFSC e maior eficiência nas trocas de dados entre os sistemas. O nodo mestre é responsável por atuar como servidor DHCP, atribuindo automaticamente endereços IP aos nodos de computação com base em seus endereços MAC. Essa abordagem simplifica a configuração e o gerenciamento da conectividade interna, eliminando a necessidade de configurações manuais.

Para a transmissão de dados e disponibilização de arquivos entre os nodos e clientes, foram configurados servidores HTTP e FTP. O protocolo HTTP, amplamente utilizado na web, permite a transferência de arquivos de forma robusta e estruturada, sendo ideal para distribuir pacotes ou documentação. Por sua vez, o protocolo FTP é empregado para transmissões rápidas e diretas de arquivos, e juntamente com o servidor DHCP e o provisionador tem a função de distribuir as imagens de OS para os nodos de computação.

A conexão remota ao cluster é realizada exclusivamente por meio do protocolo *Secure Shell* – (SSH), reconhecido por sua alta segurança na transmissão de dados. Preferencialmente, são utilizadas chaves de autenticação criptografadas, que adicionam uma camada de proteção ao substituir senhas convencionais por pares de chaves pública e privada. No entanto, para oferecer flexibilidade no acesso, o sistema também suporta autenticação baseada em senhas. Nesse caso, foi implementada a ferramenta `fail2ban`, que monitora tentativas de login e bloqueia automaticamente endereços IP após múltiplas tentativas mal-sucedidas, prevenindo ataques de força bruta. A Figura 29 ilustra a tela de login via SSH do Cluster Aqua.

Figura 29 – Tela de login do Cluster Aqua por meio do SSH



Fonte: Autor (2024).

Como medida adicional de segurança, a porta padrão do SSH foi alterada para uma não convencional, reduzindo a exposição a ataques automatizados. Além disso, o *firewall* nativo do Linux, `iptables`, foi configurado para restringir conexões na interface

de rede conectada à RedeUFSC, mantendo abertas apenas as portas estritamente necessárias para a operação do cluster. Essas configurações de conectividade e segurança asseguram a proteção do ambiente contra acessos não autorizados, ao mesmo tempo que facilitam a troca de dados e a administração do cluster.

4.2.3 Provisionamento e Escalonamento

O provisionamento e o escalonamento de tarefas no cluster foram configurados com base nas diretrizes do OpenHPC, utilizando o *Extreme Cloud Administration Toolkit* (xCAT) como ferramenta de provisionamento. O xCAT é responsável por automatizar o ciclo completo de vida dos nodos, abrangendo desde a instalação inicial até a manutenção contínua, permitindo uma configuração centralizada e consistente para todo o cluster.

A ferramenta é responsável pela criação de imagens de sistema operacional mínimas e customizadas para os nodos de computação, utilizando o módulo `genimage`. As imagens geradas incluem apenas os componentes essenciais para a execução das tarefas realizadas no NSO, reduzindo o consumo de recursos. A inicialização dos nodos é gerenciada pelo protocolo PXE, configurado no servidor DHCP do nodo mestre. O PXE permite que os nodos obtenham o sistema operacional diretamente pela rede, sem a necessidade de armazenamento local. Isso possibilita uma recuperação rápida de nodos em caso de falhas, além de simplificar a reconfiguração ou atualização do ambiente.

Para o escalonamento de tarefas, foi configurado o SLURM, organizado em três filas/partições distintas, cada uma para diferentes tipos de carga de trabalho. A fila padrão (`normal`) inclui os principais nodos do cluster (`nodo01`, `nodo02`, `nodo03`, `nodo04`) e é projetada para aplicações gerais. Uma segunda fila (`lowmem`) foi criada para processamentos que demandam baixa quantidade de memória RAM, enquanto a terceira fila (`gpu`) é destinada a tarefas que requerem o uso das GPUs disponíveis no cluster.

Os recursos disponíveis em cada fila/partição do SLURM estão detalhados na Tabela 5, que apresenta a quantidade de nodos alocados, as características de *hardware*, como núcleos de CPU, memória RAM e disponibilidade de GPUs, para cada uma das partições configuradas. No entanto, o nodo `nodo05`, que compõe a fila `gpu`, ainda está em processo de configuração, pois os drivers da NVIDIA necessários para o funcionamento das GPUs não foram instalados até o momento.

Tabela 5 – Filas de processamento configuradas no SLURM do Cluster Aqua

Fila	**Nodos**	**Núcleos***	**Memória***
`normal`	`nodo01`, `nodo02`, `nodo03`, `nodo04`	144	1024GB
`lowmem`	`nodo05`, `nodo06`, `nodo07`	32	288GB
`gpu`	`nodo05`	12	128GB

* Valores agregados de todos os recursos disponíveis na fila.

Fonte: Autor (2024).

Adicionalmente, foi implementado um banco de dados baseado no MariaDB para registrar estatísticas e informações sobre as tarefas em execução e finalizadas. Esse registro permite o monitoramento e a análise do uso de recursos, bem como a identificação de gargalos no escalonamento de tarefas, contribuindo para uma gestão eficiente do cluster.

4.3 BENCHMARK

Para avaliar o desempenho e a eficiência geral do Cluster Aqua, foi aplicado o *benchmark High Performance LINPACK* – (HPL), utilizado como métrica padrão para a classificação de supercomputadores no ranking TOP500 (2023). Este teste foi escolhido devido a sua capacidade de medir o desempenho máximo em operações de ponto flutuante, fornecendo uma referência direta da capacidade computacional do cluster.

4.3.1 High Performance LINPACK

O HPL é um *benchmark* projetado por Dongarra (1987) para resolver sistemas de equações lineares, do tipo $Ax = b$, utilizando aritmética de precisão dupla (`doubles` de 64-bits). A matriz A é densa, com elementos gerados aleatoriamente, enquanto o vetor b é configurado para produzir soluções exatas predefinidas. O *benchmark* utiliza a decomposição LU com pivotamento parcial para a resolução do sistema, empregando bibliotecas otimizadas de álgebra linear e MPI para comunicação entre processos.

Os principais parâmetros do HPL incluem o tamanho da matriz quadrada (N), o bloco de particionamento (NB) e o número de processos na dimensão vertical e horizontal da grade bidimensional (P e Q, respectivamente). Um valor maior para N geralmente resulta em um teste com melhores resultados, pois aumenta a razão entre operações computacionais e comunicações. No entanto, N está limitado pela quantidade de memória disponível nos nodos. O tamanho ideal da matriz A, conforme a memória disponível no sistema, pode ser estimado a partir da Equação 12, onde α é um fator corretivo de segurança no intervalo $[0,1]$, usualmente próximo a $0{,}9$.

$$N \approx \alpha \sqrt{\frac{\#\text{nodos} \times \text{RAM [GB]} \times 1024^3}{\texttt{sizeof(double)}}} \tag{12}$$

O tamanho do bloco de particionamento NB é um parâmetro dependente do modelo de processador utilizado no teste e é fornecido pelo fabricante do *hardware*. Determinar valores balanceados de P e Q envolve uma escolha que iguale seu produto pelo número total de processos disponíveis, *i.e.*, $P \times Q = \#$ processos, tal que $Q \neq P$ e Q seja ligeiramente maior que P (Dongarra, 1987).

No teste realizado no Cluster Aqua foram usados os três nodos operantes da fila `normal` – `nodo01`, `nodo02` e `nodo03`. Cada um dos nodos possui dois processadores Intel

Xeon E5-2630v3@2.4GHz com *Hyperthreading* habilitado[1] e 256GB de memória RAM. Esta configuração permite a execução do HPL em 96 processos paralelos, o que resulta em valores de P e Q iguais a 8 e 12, respectivamente. Por meio da Equação 12 e considerando $\alpha = 0.9$ determinou-se um tamanho da matriz $N = 280.000$, como demostrado abaixo:

$$N \approx 0{,}9 \sqrt{\frac{3 \times 256 \times 1024^3}{8}} \approx 288.953$$

para processadores da arquitetura Haswell – como os empregados neste teste – o valor mais adequado para NB é 192 (Dolbeau, 2018). A Tabela 6 sumariza os parâmetros usados para execução do HPL.

Tabela 6 – Parâmetros utilizados no benchmark HPL

# processos	**Memória**	P	Q	NB	N
96	768GB	8	12	192	280.000

Fonte: Autor (2024).

Retomando a Equação 1, que determina o desempenho máximo teórico do sistema (R_{peak}), e aplicando-a na fila de processamento usada para realização do *benchmark*, obtém-se que o R_{peak} esperado para o sistema é de $1{,}843$TFlops/s, conforme detalhado abaixo:

$$R_{peak} = 6 \times 8 \times 2{,}4 \cdot 10^9 \times 16 = 1{,}843 \cdot 10^{12} \text{ Flops/s}$$

O tempo total de execução do HPL foi de aproximadamente 3h e 40min, tendo como resultado o desempenho máximo de $R_{max} = 1{,}106$TFlops/s. Este valor de R_{max} implica em uma eficiência geral do sistema de $\eta_{sist} = 60{,}01\%$, como demonstrado na Tabela 7.

Tabela 7 – Resultados do benchmark HPL

R_{peak}	R_{max}	η_{sist}
1,843TFlops/s	1,106TFlops/s	60,01%

Fonte: Autor (2024).

Os resultados obtidos são indicadores positivos para o Cluster Aqua. O valor de η_{sist} está alinhado com a média de eficiência observada em sistemas listados no ranking TOP500 (2023), e acima do melhor colocado brasileiro que tem uma eficiência de 45%

[1] O *Hyperthreading* é uma tecnologia presente em alguns processadores da Intel que permite que um único núcleo execute duas *threads* simultaneamente.

(Tabela 3). Este fator é especialmente relevante considerando que o cluster é composto por *hardware commodity* e interconexões convencionais. Embora a eficiência não alcance o valor teórico máximo devido às limitações inerentes, os resultados destacam a capacidade do Cluster Aqua em oferecer recursos adequados para aplicações de alta performance em seu escopo de utilização.

O desempenho alcançado pelo Cluster Aqua pode ainda ser colocado em uma perspectiva histórica. Em 1997, o supercomputador ASCI Red tornou-se o primeiro sistema a atingir a marca de 1TFlops/s no teste HPL, utilizando 9.298 processadores Pentium Pro e ocupando a primeira posição no ranking TOP500 (2023) até o ano de 2001. Na época, esse feito representava um marco tecnológico significativo, destacando o avanço da computação paralela e distribuída[2] (Sandia National Laboratories, 2006). Comparativamente, o Cluster Aqua, equipara-se ao ASCI Red em desempenho máximo alcançado no HPL. Além disso, considerando a evolução histórica da lista TOP500, a performance do Cluster Aqua permitiria que o sistema figurasse entre os 500 melhores até o ano de 2006.

4.4 CONSIDERAÇÕES DO CAPÍTULO

Este capítulo apresentou uma visão abrangente da infraestrutura implantada, destacando as soluções adotadas para a construção de um ambiente de computação de alto desempenho eficiente e funcional no NSO. A implantação do *hardware* foi detalhada, abordando os dispositivos utilizados, a topologia de rede em estrela e as conexões estabelecidas entre os nodos. Aspectos relacionados à configuração de *software* também foram discutidos, incluindo a escolha do sistema operacional, a implementação do sistema de arquivos, medidas de segurança, estratégias de conectividade, ferramentas de provisionamento e mecanismos de escalonamento de tarefas, com justificativas para cada decisão tomada.

Por fim, uma avaliação do desempenho do Cluster Aqua foi conduzida utilizando o *benchmark* HPL, que permitiu mensurar o desempenho do sistema frente a tarefas intensivas em cálculo numérico. A configuração dos parâmetros do *benchmark* foi analisada com atenção, evidenciando o impacto dessas escolhas na eficiência computacional. Mesmo com limitações impostas pelo número de nodos disponíveis durante os testes, os resultados obtidos confirmaram a capacidade do cluster de atender às demandas computacionais propostas, oferecendo uma base sólida para futuras expansões e melhorias.

[2] Recentemente, o novo marco do processamento Exaflop (*i.e.*, 10^{18} Flops/s) foi atingido pelo supercomputador Frontier, do Oak Ridge National Laboratory (2022)

5 SYNAPSE MEETS SLURM:

PROPOSTA E AVALIAÇÃO DE UM MIDDLEWARE PARA PARALELIZAÇÃO DE ALGORITMOS DE OTIMIZAÇÃO POPULACIONAIS

A principal atividade desenvolvida no NSO consiste na otimização de sistemas de engenharia utilizando o *software* Synapse (2023), ferramenta especializada desenvolvida no laboratório. Esse processo requer a execução de modelos computacionais, frequentemente associados a altos custos computacionais devido ao volume e a complexidade de simulações envolvidas. Nesse contexto, o Cluster Aqua, com sua capacidade de processamento, possui o potencial de acelerar essas otimizações, permitindo a distribuição das simulações de possíveis soluções.

Este capítulo apresenta uma proposta de *middleware* para integrar o Synapse a clusters gerenciados pelo SLURM, viabilizando a paralelização das tarefas de otimização e explorando o potencial do ambiente de computação de alto desempenho disponível. A proposta abrange o desenvolvimento de mecanismos que conectam as etapas do Synapse aos recursos de gerenciamento e escalonamento do SLURM, buscando não apenas melhorar o desempenho das otimizações, mas também garantir uma integração eficiente e transparente entre o *software* e a infraestrutura computacional. O ganho de desempenho fornecido pelo processamento distribuído é avaliado por meio de métricas de escalabilidade.

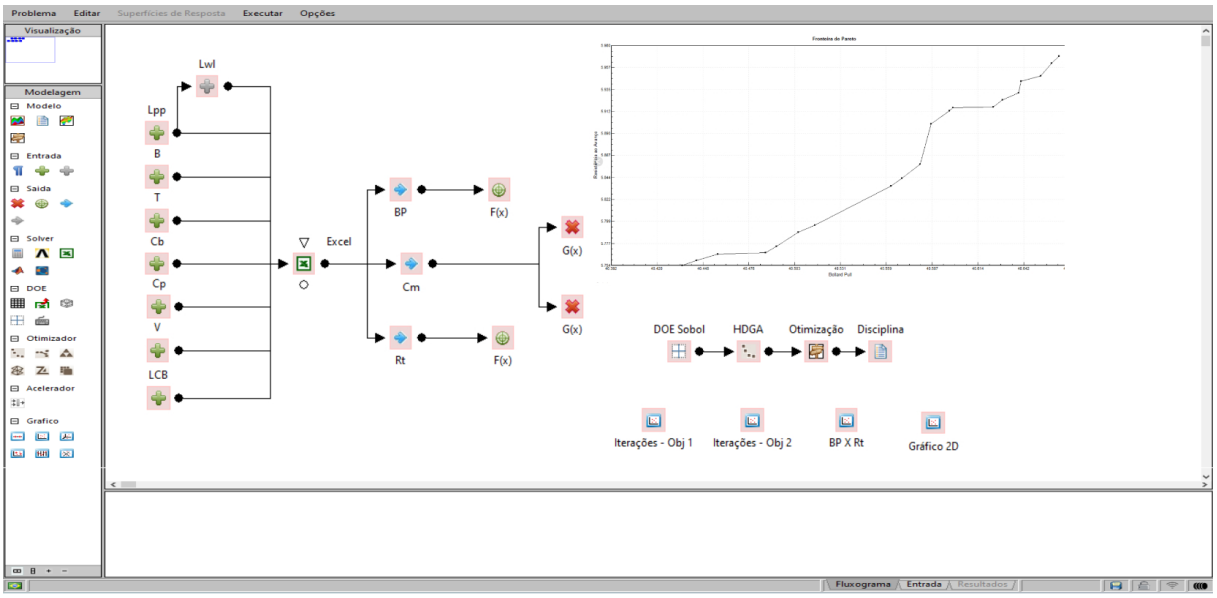
5.1 SYNAPSE ENGENHARIA MULTIDISCIPLINAR

O Synapse Engenharia Multidisciplinar é um *software* de otimização de projetos desenvolvido no NSO em colaboração com a indústria, com o objetivo de atender às demandas específicas de engenharia. A ferramenta destaca-se por oferecer um ambiente colaborativo e amigável ao usuário, facilitando a modelagem de problemas de otimização de forma visual, utilizando fluxogramas, como ilustrado na Figura 30. Essa abordagem intuitiva reduz a complexidade associada à configuração de problemas de otimização, ampliando sua acessibilidade para usuários com diferentes níveis de experiência (Synapse, 2023).

Além de sua interface intuitiva, o Synapse (2023) possui integração com diferentes ferramentas de Planejamento de Experimentos (*Design of Experiments* – DoE) e análise, incluindo AnSYS APDL, Python, Excel e MATLAB. Essa versatilidade permite que o *software* seja adaptado a diferentes projetos e integre diferentes disciplinar no processo de otimização.

Entre os métodos de otimização disponíveis no Synapse, os meta-heurísticos populacionais, como os algoritmos genéticos, são particularmente relevantes para este trabalho. Esses métodos permitem a exploração de soluções de problemas de engenharia mono

Figura 30 – Modelagem de problema de otimização por fluxogramas no Synapse



Fonte: Synapse (2023).

e multiobjetivos, viabilizando a busca por projetos otimizados de forma robusta, além de serem altamente paralelizáveis. A seguir são discutidos alguns dos aspectos fundamentais dos métodos e algoritmos de otimização empregados no *software*.

5.1.1 Otimização

Segundo Sobieszczanski-Sobieski *et al.* (2017), métodos de otimização são fundamentais tanto em sistemas naturais quanto artificiais. Na natureza, a otimização é evidente em processos como a seleção natural e os estados de mínima energia. Da mesma forma, em sistemas desenvolvidos pelo ser humano, a otimização apoia uma melhor tomada de decisão ao fornecer soluções e *trade-offs* ótimos – porém nem sempre globais[1] – o que é essencial em estratégias de negócios, projetos de engenharia e políticas públicas.

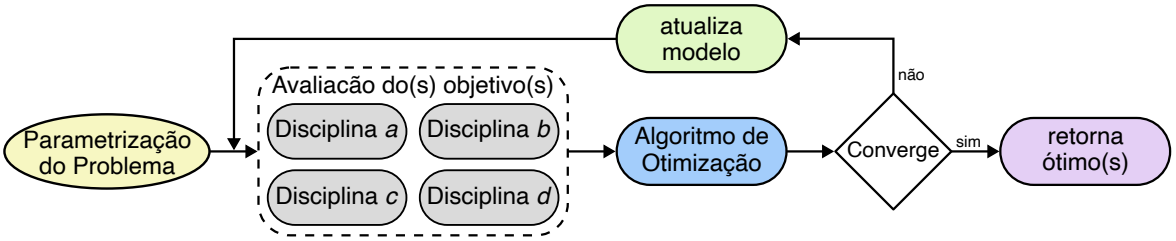
Formalmente, a otimização é definida como o processo sistemático de identificar e ajustar parâmetros para atingir os valores máximos ou mínimos de certos aspectos quantificáveis de um sistema. Esse processo pode levar em consideração tanto restrições explícitas quanto implícitas, garantindo que a solução atenda às limitações e requisitos predefinidos, conforme demonstrado na Equação 13, onde $\vec{x}$ é o vetor de variáveis ou parâmetros, $f(\vec{x})$ é a função objetivo e C é um conjunto de funções escalares de $\vec{x}$ que impõem as restrições no domínio de soluções viáveis (Nocedal; Wright, 2006).

[1] ótimos globais representam a melhor solução em todo o espaço de busca, enquanto ótimos locais correspondem a soluções que minimizam ou maximizam a função objetivo em uma região limitada do espaço de busca. Em processos de otimização, não há garantia de alcançar a otimalidade global, especialmente em problemas não convexos ou com múltiplos extremos (Nocedal; Wright, 2006).

$$\begin{aligned} \min_{\vec{x}\in\mathbb{R}^n} \quad & f(\vec{x}) \\ \text{sujeito a} \quad & C = \begin{cases} c_i(\vec{x}) = h_i \\ c_j(\vec{x}) \leq g_j \end{cases} \end{aligned} \tag{13}$$

No contexto de problemas de engenharia, a Otimização Multidisciplinar de Projetos (*Multidisciplinary Design Optimization* – MDO) é frequentemente empregada. A MDO oferece uma estrutura para abordar sistemas de engenharia complexos ao integrar várias disciplinas interativas e ferramentas de análise em um único processo de otimização. Essa perspectiva holística dos *frameworks* MDO permite uma abordagem sistemática em que diversos aspectos são otimizados em conjunto, garantindo que o projeto final equilibre diferentes objetivos concorrentes (Tancredi, 2009). A Figura 31 demonstra o fluxo típico de um processo de MDO.

Figura 31 – Fluxo de trabalho de um problema de otimização integrando ferramentas de análise multidisciplinar no processo



Fonte: Autor (2024).

Problemas de otimização podem ser classificados como mono ou multiobjetivo, dependendo do número de critérios que se busca otimizar. Na otimização mono-objetivo, a solução ótima maximiza ou minimiza uma única função objetivo, geralmente com base em um critério bem definido. Por outro lado, problemas multiobjetivo envolvem múltiplos critérios que, em muitos casos, são conflitantes entre si. Nesses casos, não existe uma única solução ótima, mas sim um conjunto de soluções conhecidas como a Fronteira de Pareto, onde melhorias em um objetivo podem implicar em sacrifícios em outro. A abordagem multiobjetivo é amplamente utilizada na otimização multidisciplinar, pois reflete a realidade de projetos complexos, nos quais diferentes aspectos (*e.g.*, custo, desempenho, tempo, sustentabilidade) precisam ser equilibrados para atender às demandas de projeto (Sobieszczanski-Sobieski *et al.*, 2017).

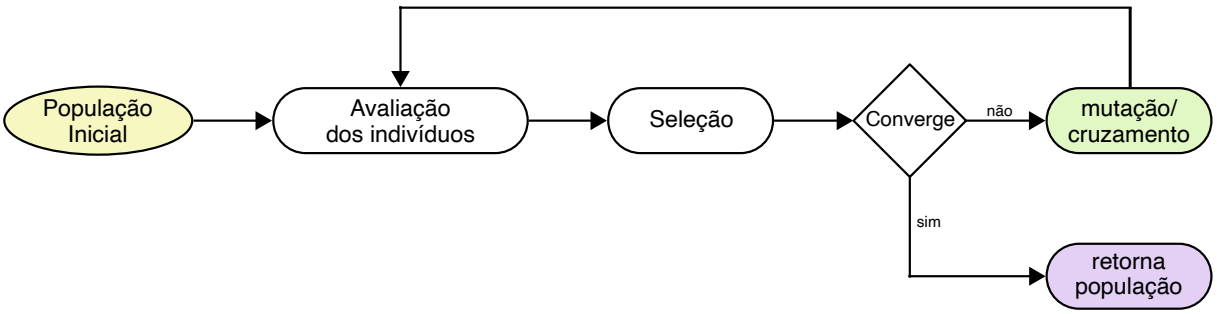
Independentemente da abordagem, a otimização é inerentemente iterativa, começando com uma estimativa inicial dos valores ótimos e gerando uma sequência de estimativas progressivamente melhores até alcançar uma solução. A estratégia empregada para transitar de uma iteração para a próxima diferencia os vários algoritmos. A maioria dos métodos utiliza os valores da função objetivo $f(\vec{x})$, as restrições C e, possivelmente, suas

derivadas. Enquanto alguns algoritmos acumulam informações de iterações anteriores, outros dependem apenas de informações locais do ponto atual (Nocedal; Wright, 2006).

No contexto deste trabalho, as meta-heurísticas populacionais como os Algoritmos Genéticos (*Genetic Algorithms* – GAs) são de especial interesse. Os GAs são uma classe de algoritmos de otimização inspirados nos princípios da seleção natural e genética. Eles são particularmente adequados para resolver problemas complexos e não lineares com grandes espaços de busca, onde métodos tradicionais de otimização podem enfrentar dificuldades (Holland, 1992). GAs funcionam gerando uma população de soluções potenciais, chamadas de indivíduos, que evoluem ao longo de iterações sucessivas, ou gerações. Conforme ilustrado na Figura 32, os indivíduos com melhor desempenho são selecionados para produzir a próxima geração por meio de operações genéticas como cruzamento e mutação. Esse processo continua iterativamente, guiando o algoritmo em direção a soluções melhores enquanto explora regiões diversas do espaço de projeto (Holland, 1992).

Uma das principais vantagens dos algoritmos genéticos é a capacidade de evitar ótimos locais, um problema comum em otimizações complexas. Ao manter uma população diversa e introduzir mutações aleatórias, os GAs exploram uma gama mais ampla do espaço de soluções. Isso os torna ideais para problemas de MDO, onde parâmetros interdependentes devem ser considerados (Holland, 1992). Além disso, segundo Tancredi (2009), os GAs podem ser paralelizados de forma eficiente, permitindo que diferentes partes do espaço de soluções sejam exploradas simultaneamente em sistemas de computação distribuída, acelerando significativamente o processo de otimização e possibilitando a solução de problemas que seriam computacionalmente inviáveis usando métodos tradicionais.

Figura 32 – Fluxograma de um Algoritmo Genético



Fonte: Autor (2024).

5.2 DESENVOLVIMENTO DO MIDDLEWARE

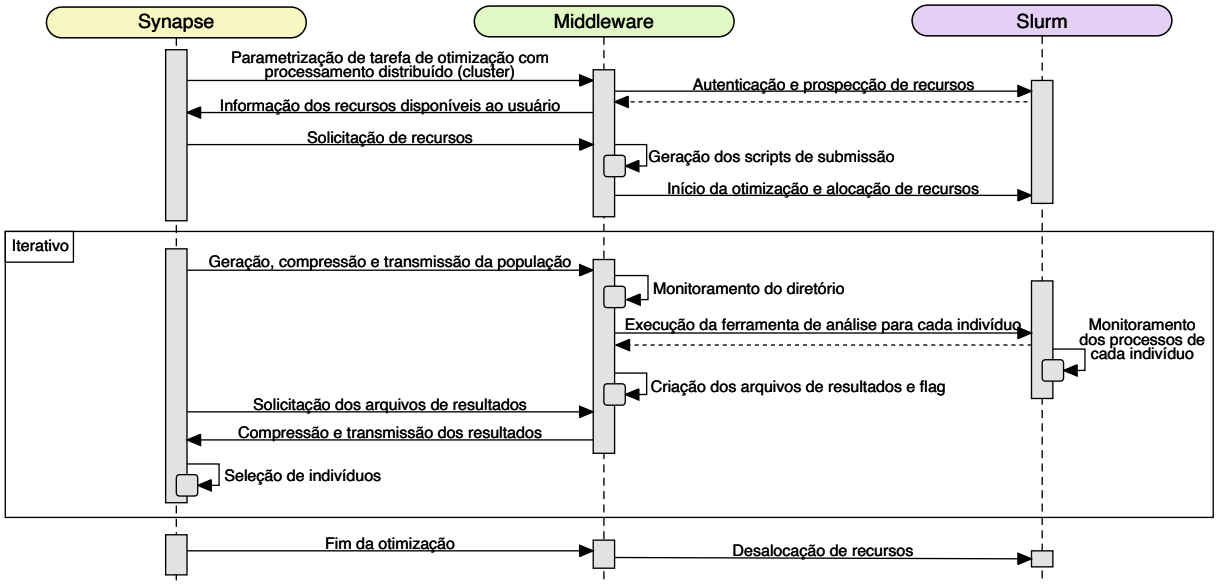
A fim de expandir as capacidades do *software* Synapse para ambientes de computação de alto desempenho, propõe-se o desenvolvimento de um *middleware*, denominado *Synapse meets Slurm* (SmS). O SmS possibilita a integração do Synapse com sistemas HPC gerenciados pelo SLURM, permitindo a execução de tarefas de otimização de forma

distribuída em clusters. Essa abordagem visa superar a limitação do Synapse, atualmente restrito a Sistemas Operacionais Windows e projetado em Delphi, sem necessidade de portá-lo integralmente para plataformas GNU/Linux. Com o uso do SmS, espera-se que o processamento distribuído, ao aumentar a taxa de soluções examinadas durante o processo de otimização, contribua com a otimização ao reduzir seu tempo total de execução.

5.2.1 Arquitetura e Funcionalidade

Para minimizar alterações no Synapse e ao mesmo tempo dar suporte a processamento em clusters, um *middleware* foi concebido e implementado. A implementação usa *bourne again shell* (bash), em sua maior parte, com uma pequena porção de código desenvolvida em PowerShell (rotinas de autenticação, comunicação e transmissão de dados entre o *host* onde o Synapse está em execução e o cluster). A Figura 33 apresenta um diagrama de sequência, e ilustra o fluxo de interação entre o Synapse e o cluster, detalhando como o *middleware* gerencia a comunicação e os recursos alocados. Toda comunicação entre o Synapse e o cluster ocorre via *ssh*, sendo, então, interpretada pelo *middleware* de forma transparente ao usuário.

Figura 33 – Diagrama de sequência do middleware proposto



Fonte: Gabardo *et al.* (2024).

De forma breve, ao início de um processo de otimização, o *middleware* realiza a autenticação no cluster, o qual informa os recursos disponíveis ao usuário. Em seguida um *job* é submetido pelo Synapse ao SLURM. O *job* do *middleware* permanece em execução até o fim do processo de otimização usando os recursos requisitados de maneira exclusiva, sendo responsável por: (i) receber arquivos contendo indivíduos (soluções) que devem ser avaliados; (ii) alocar recursos adicionais e distribuir os indivíduos de acordo com a quantidade de recursos solicitada; (iii) monitorar e gerenciar uma fila interna ao

middleware de indivíduos, realizando balanceamento de carga; (iv) compactar arquivos de resultado de uma geração do algoritmo genético e escrever arquivos de *flag*, indicando que os resultados estão prontos e que o Synapse pode transferi-los a partir do cluster.

Os scripts de submissão em lote são gerados automaticamente para cada um dos nodos solicitados, e a requisição de alocação dos recursos é feita ao SLURM por meio do comando `sbatch`. Como a alocação é feita individualmente para cada nodo, a análise dos indivíduos pode ocorrer mesmo com um número de nodos disponíveis inferior ao requerido. Após a alocação dos nodos, um diretório no caminho especificado pelo usuário é criado para armazenar os dados necessários para as simulações e seus resultados. Os *jobs* do *middleware* em execução nos nodos de computação monitoram constantemente as mudanças realizadas no diretório e informam a quantidade de núcleos de processamento disponíveis.

5.2.2 Modelo Mestre-Trabalhador

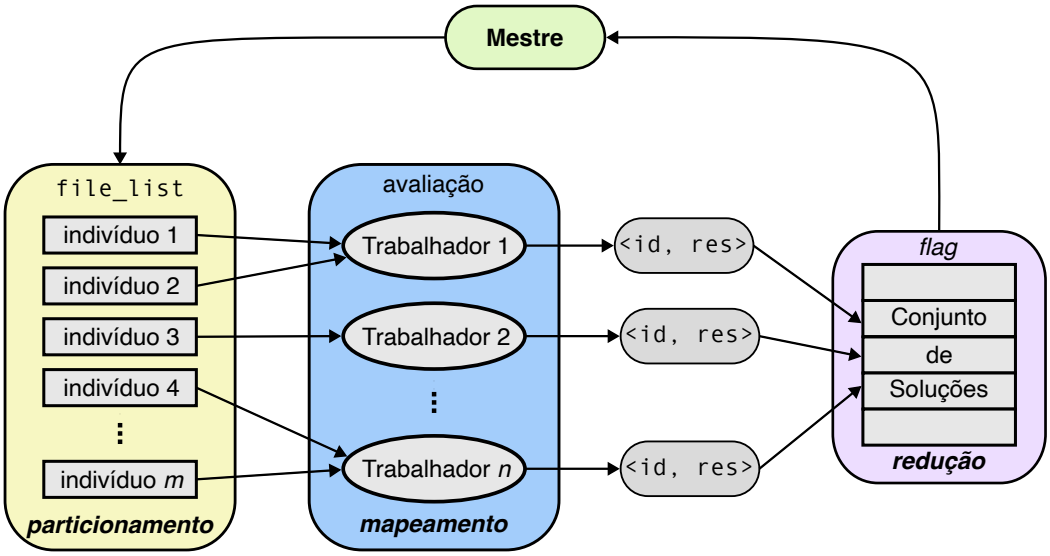
O *middleware* desenvolvido adota uma abordagem baseada no modelo mestre-trabalhador. Neste modelo, o nodo mestre coordena o processo de otimização, dividindo a população em subgrupos e distribuindo-os para avaliação nos nodos de trabalho do cluster. Cada nodo de trabalho avalia os indivíduos designados, enviando os resultados de volta ao nodo mestre, que realiza a agregação e processamento dessas informações. Essa estrutura é análoga ao padrão *MapReduce*, onde a fase de mapeamento corresponde à avaliação paralela dos indivíduos em diferentes nodos, enquanto a fase de redução centraliza os resultados para aplicar operadores genéticos, como seleção, mutação e recombinação.

Os indivíduos de uma população são gerados no computador do usuário por um dos métodos de DoE disponíveis no Synapse. Estes são então compactados e transmitidos para o diretório da simulação no cluster via `scp`, juntamente com um identificador do número da geração. A chegada deste arquivo ao cluster indica que há uma nova geração a ser avaliada. Após a extração dos indivíduos o arquivo `file_list` é criado, contendo em cada uma de suas linhas, uma fila de indivíduos a ser avaliada em cada nodo. A geração e balanceamento de carga destas filas são realizados pelo *middleware*. Como o custo computacional da avaliação dos indivíduos em uma mesma geração é similar, a população é dividida de maneira proporcional ao número de núcleos de cada um dos nodos. Os processos são distribuídos e executados como determinado no arquivo `file_list`, e a quantidade de processos simultâneos é monitorada por meio da ferramenta *process status* (`ps`). Quando o tamanho da população é inferior ao número de núcleos, múltiplas *threads* podem ser usados para um mesmo indivíduo, caso haja suporte por parte da ferramenta de análise.

Completada a simulação de todos os indivíduos, um arquivo de *flag* é criado. O Synapse realiza requisições periódicas para verificar a existência deste arquivo, assim que identificado, os resultados são compactados e transmitidos para o *host* do Synapse, onde

são comparados e selecionados. A seguir, uma nova população é gerada, e o procedimento descrito ocorre de maneira iterativa até que o espaço de busca seja esgotado, um conjunto de soluções ótimas encontrado e o processo de otimização seja encerrado. Por fim, um arquivo sinalizando o término da otimização encerra os trabalhos alocados no cluster. A Figura 34 apresenta o fluxo de distribuição das análises realizado pelo SmS.

Figura 34 – Modelo mestre-trabalhador aplicado as análises do Synapse



Fonte: Autor (2024).

5.2.3 Tolerância a Falhas

A robustez do processo de otimização é garantida por mecanismos de tolerância a falhas integrados ao SmS e ao Synapse. A natureza iterativa dos algoritmos genéticos, em que as soluções evoluem geração após geração, permite tratar cada geração como um ponto de recuperação (*checkpoint*). Dessa forma, falhas que ocorram durante a execução de uma geração podem ser isoladas e tratadas sem comprometer o progresso global da otimização. O uso de *checkpoints* evita que o processo de otimização tenha que ser reiniciado desde o início, reduzindo o impacto das falhas no tempo total de execução.

No nível dos indivíduos, quaisquer falhas detectadas durante a avaliação são tratadas com o descarte dessas soluções. Isso impede que dados inválidos interfiram no processo de seleção ou na evolução da população. Para falhas em larga escala, como a interrupção completa de uma geração ou a indisponibilidade de múltiplos nodos, o SmS realiza a ressubmissão dos *jobs*. A geração em questão é reiniciada, garantindo que todos os indivíduos sejam avaliados corretamente antes de prosseguir para a próxima etapa.

5.3 VALIDAÇÃO

O *middleware* proposto foi implementado e integrado à última versão do Synapse Offshore, uma customização do Synapse destinada à otimização de sistemas de ancora-

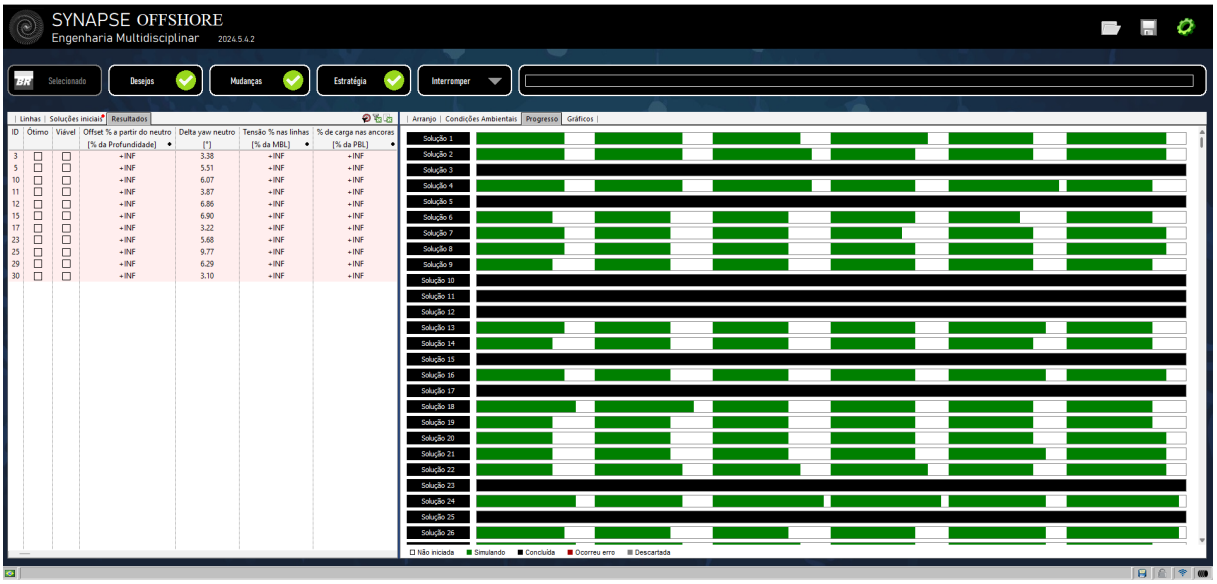
gem de plataformas flutuantes de produção de óleo e gás. Os testes foram realizados no Cluster Aqua e validaram a arquitetura proposta, que tem sido utilizada com sucesso nos projetos de pesquisa do NSO. Nesta seção, são apresentados os testes realizados e os resultados obtidos em termos de ganho de desempenho e escalabilidade, demonstrando a redução no tempo total de execução do processo de otimização proporcionada pelo SmS.

5.3.1 Testes

Os testes consistiram na execução de um processo de otimização aplicado a uma plataforma de petróleo do tipo semissubmersível, com o objetivo de maximizar seu deslocamento na direção norte, enquanto restringia os deslocamentos em outras direções e a tensão máxima calculada nas linhas de ancoragem. As variáveis do problema incluem parâmetros geométricos das linhas, que foram ajustados ao longo das gerações para atender aos critérios de otimização estabelecidos. O tamanho de população inicial usada no GA foi de 30 indivíduos, com taxa de mutação de 80% e um máximo de 100 gerações foi estabelecida como critério de parada. Para avaliar o ganho de desempenho e a escalabilidade do SmS, o processo foi executado utilizando configurações que variaram entre 1, 4, 8, 16, 30, 60, 90 e 120 núcleos de processamento disponíveis no Cluster Aqua.

A Figura 35 apresenta uma captura de tela do Synapse Offshore durante um dos testes. A direita da interface, barras verdes representam os processos distribuídos no cluster. Cada linha corresponde a um indivíduo sendo avaliado, enquanto as colunas indicam paralelização aplicado à análise de um único indivíduo. Essa organização evidencia os diferentes níveis de granularidade do paralelismo explorados pelo SmS, incluindo tanto a distribuição entre nodos quanto a paralelização interna de ferramentas de análise.

Figura 35 – Tela de monitoramento dos processos no Synapse

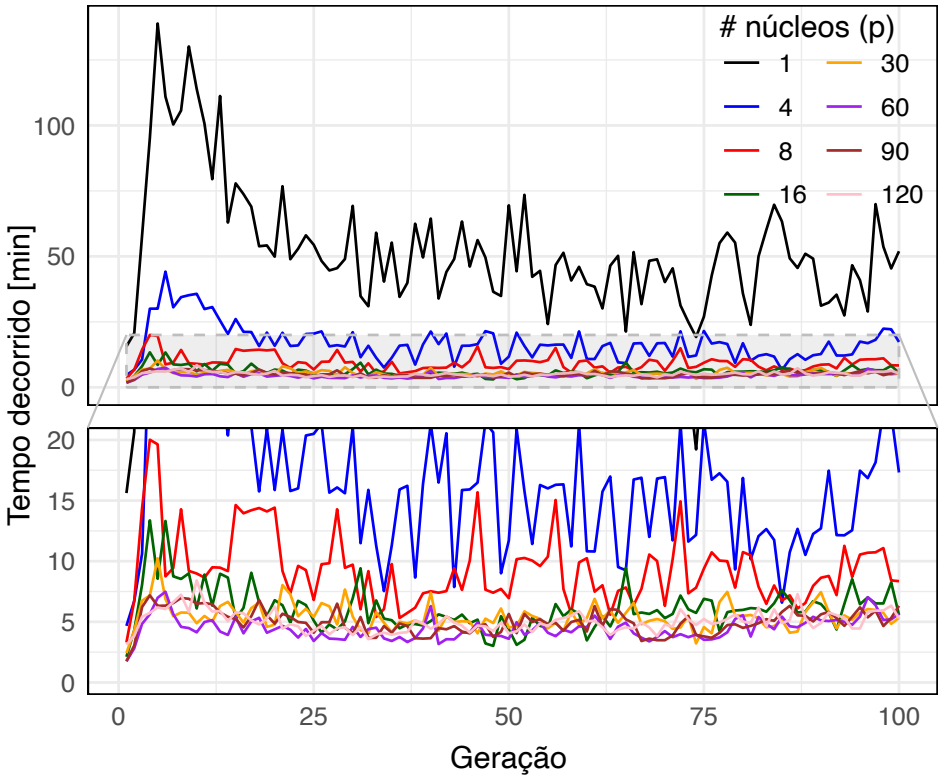


Fonte: Autor (2024).

5.3.2 Ganho de Desempenho e Escalabilidade

A Figura 36 apresenta a evolução do tempo entre gerações do processo de otimização em função do número de núcleos (p) utilizados. Observa-se uma redução significativa no tempo entre gerações ao aumentar p, especialmente para configurações de até 60 núcleos. Esse comportamento evidencia a eficiência do *middleware* na distribuição das análises, permitindo maior paralelismo na avaliação dos indivíduos. No entanto, acima de 60 núcleos, a redução no tempo por geração estabiliza, indicando uma saturação no aproveitamento dos recursos adicionais.

Figura 36 – Tempo entre gerações do processo de otimização



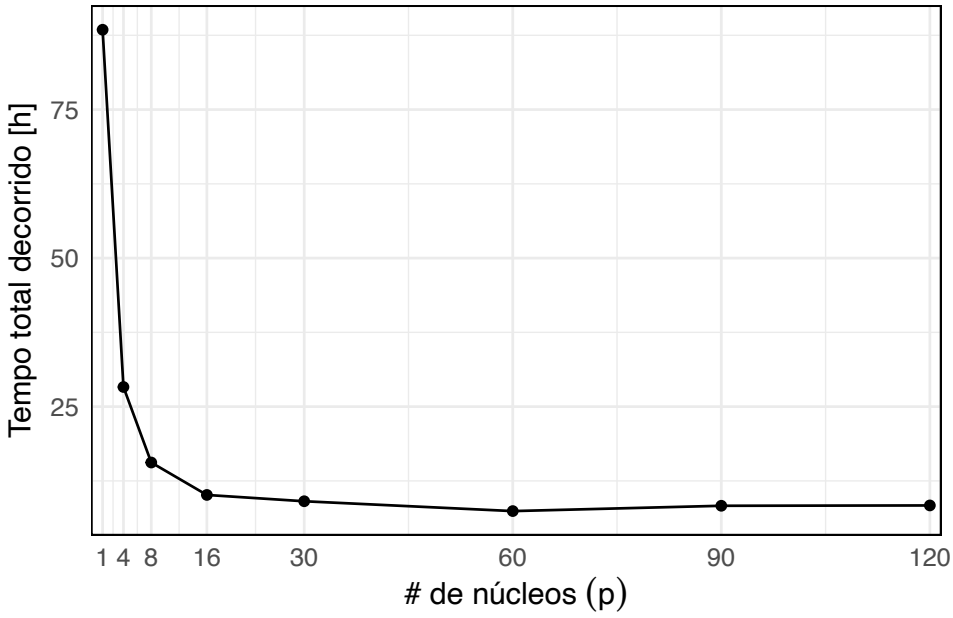
Fonte: Autor (2024).

A Figura 37 ilustra o tempo total decorrido do processo de otimização para diferentes números de núcleos. O tempo total apresenta uma tendência de redução até 60 núcleos, atingindo o menor valor nessa configuração. Contudo, observa-se um aumento no tempo total para 90 e 120 núcleos, refletindo o impacto dos limites de contenção e coerência.

O ganho de desempenho (S_p) obtido com o aumento de p é apresentado na Figura 38. O *speedup* segue um crescimento real para valores de $p \leq 60$. Para valores de $p \geq 30$, o aumento é mais gradual, devido a limitações inerentes ao paralelismo, como predito pelos modelos teóricos de escalabilidade. O melhor ajuste dos resultados experimentais ao modelo USL pode ser atribuído à abordagem adotada, que paraleliza as tarefas tanto ao nível

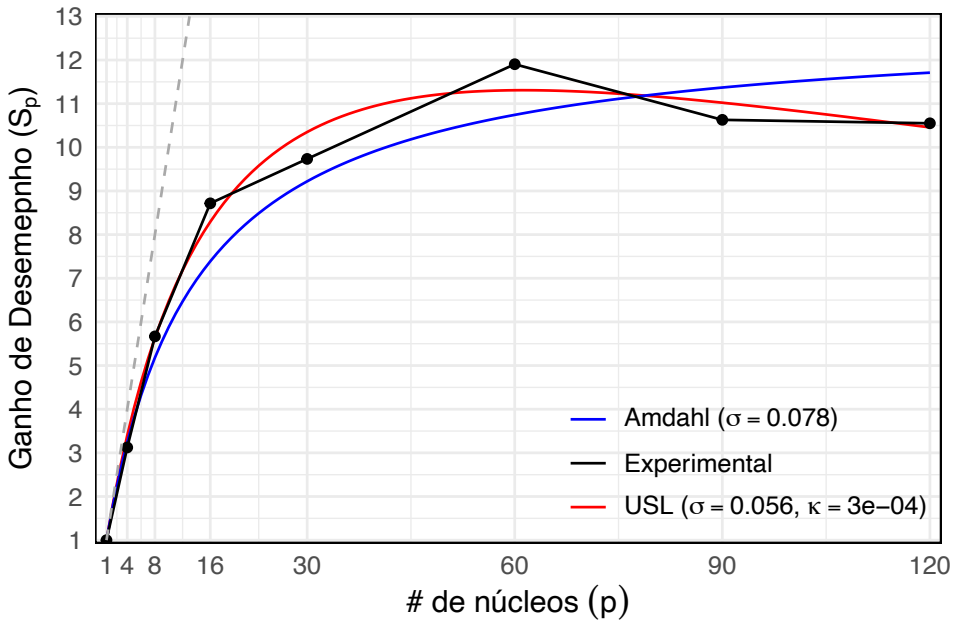
de indivíduos quanto ao nível da ferramenta de análise. Essa estratégia introduz restrições adicionais, como a necessidade de coerência na leitura e escrita de arquivos de resultados compartilhados entre diferentes processos, representadas pelo termo κ na Equação 9. Esse comportamento ressalta o impacto dos limites de escalabilidade, especialmente à medida que mais núcleos são adicionados.

Figura 37 – Tempo total decorrido do processo de otimização



Fonte: Autor (2024).

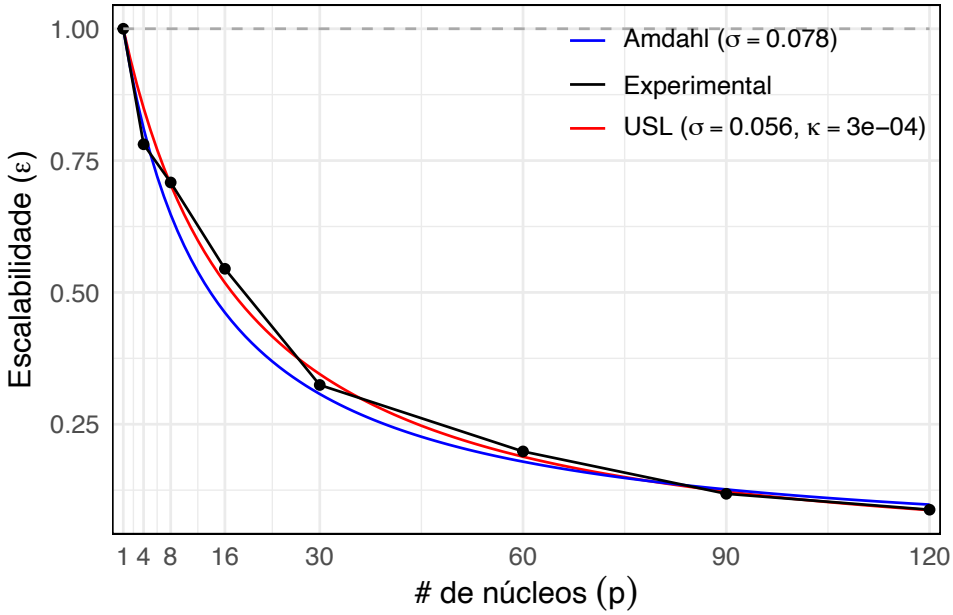
Figura 38 – Ganho de desempenho do processo de otimização



Fonte: Autor (2024).

A escalabilidade do *middleware*, avaliada experimentalmente e comparada aos modelos teóricos, é demonstrada na Figura 39. A proximidade entre os dados experimentais e os modelos reforça a validade da abordagem empregada, especialmente para valores de $p \leq 60$. A partir desse ponto, a baixa escalabilidade reflete os gargalos observados nas outras métricas, destacando os limites práticos da solução.

Figura 39 – Escalabilidade do processo de otimização



Fonte: Autor (2024).

Por fim, a Tabela 8 sintetiza os principais resultados dos testes, apresentando métricas como o tempo total do processo de otimização (t_{tot}), a mediana e o desvio padrão do tempo entre gerações ($med(t)$ e $\sigma(t)$), o ganho de desempenho (S_p) e a escalabilidade (ε). Esses dados corroboram as observações feitas nas Figuras, evidenciando que o *middleware* foi capaz de acelerar consideravelmente a otimização, com um valor ótimo de núcleos alocados $p^* = 60$, para este problema especifico.

Tabela 8 – Resultados dos testes de ganho de desempenho do middleware

p	t_{tot}[min]	$med(t)$[min]	$\sigma(t)$[min]	S_p	ε
1	5.306,25	49,00	23,40	–	–
4	1.698,42	16,20	6,88	3,12	0,78
8	935,70	9,18	2,94	5,67	0,71
16	608,17	5,78	1,80	8,72	0,55
30	544,53	5,31	1,09	9,74	0,32
60	445,22	4,37	0,90	11,92	0,20
90	498,85	4,94	0,99	10,64	0,12
120	502,22	4,90	0,86	10,57	0,09

Fonte: Autor (2024).

5.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo, foi apresentado o desenvolvimento e a avaliação do *middleware* SmS, projetado para integrar o *software* Synapse a clusters de computação de alto desempenho gerenciados pelo SLURM. O *middleware* foi concebido com o objetivo de expandir as capacidades do Synapse, permitindo a execução de processos de otimização em ambientes distribuídos, sem a necessidade de modificar significativamente sua arquitetura original.

Inicialmente, foi descrita a proposta do *middleware*, ressaltando sua implementação em `bash` e PowerShell, com comunicação baseada no protocolo SSH. Essa abordagem assegura a integração entre o Synapse, executado em sistemas Windows, e clusters HPC baseados em Linux, utilizando um modelo mestre-trabalhador. O funcionamento do *middleware* foi detalhado, desde a submissão de *jobs* até o gerenciamento de filas internas e o balanceamento de carga, garantindo que a execução das análises seja distribuída de forma eficiente entre os recursos disponíveis.

Também foram discutidos os mecanismos de tolerância a falhas implementados no SmS, destacando como a natureza iterativa dos algoritmos genéticos possibilita a recuperação do processo em caso de interrupções. O uso de *checkpoints* em cada geração e a ressubmissão de *jobs* garantem a robustez e continuidade do processo de otimização, mesmo diante de falhas em larga escala ou indisponibilidade de recursos.

A validação do *middleware* foi realizada por meio de sua integração ao Synapse Offshore, em aplicações práticas relacionadas à otimização de sistemas de ancoragem de plataformas flutuantes. Os testes conduzidos no Cluster Aqua demonstraram a eficiência e escalabilidade do *middleware*, com resultados detalhados nas métricas de tempo entre gerações, tempo total de execução, ganho de desempenho e eficiência. As análises evidenciaram um ponto ótimo de $p^* = 60$ núcleos alocados, além de confirmar a proximidade dos resultados experimentais aos modelos teóricos de escalabilidade, como a Lei Universal da Escalabilidade.

6 CONCLUSÃO

Este trabalho abordou o desenvolvimento e a implementação de soluções para desafios de performance e programação em Computação de Alto Desempenho (HPC). O foco principal foi a construção de uma infraestrutura capaz de suportar modelos e simulações computacionais avançados no contexto de projetos de engenharia, bem como o desenvolvimento de um *middleware* para otimização distribuída.

Inicialmente, o trabalho detalhou os fundamentos das arquiteturas HPC, destacando os conceitos de processamento paralelo e distribuído, suas diferentes arquiteturas de memória, topologias de interconexão e os componentes necessários para gerenciar sistemas distribuídos. Essa base teórica forneceu subsídios para a implementação do Cluster Aqua, um *Commodity Cluster* configurado no NSO, utilizando recursos preexistentes e tecnologias de código aberto. O Cluster Aqua foi validado com o *benchmark* HPL, demonstrando sua capacidade de atender demandas computacionais intensivas com eficiência e escalabilidade.

No âmbito do desenvolvimento para sistemas HPC, o trabalho explorou modelos de programação paralela e métricas de desempenho, com ênfase na eficiência, escalabilidade e tolerância a falhas. Esses conceitos foram aplicados no *middleware* SmS, projetado para integrar o *software* Synapse a clusters gerenciados pelo SLURM. A solução permitiu que algoritmos populacionais executados pelo Synapse Offshore fossem distribuídos de forma eficiente, aproveitando os recursos do Cluster Aqua sem a necessidade de reestruturar o *software* original.

Os testes realizados demonstraram o impacto positivo do SmS na redução do tempo total de execução de processos de otimização. Os resultados alcançados reforçam a importância da integração entre infraestrutura HPC e ferramentas especializadas como o Synapse para solucionar problemas de engenharia. Este trabalho contribuiu para ampliar a capacidade computacional do NSO, criando uma base sólida para o desenvolvimento de novos projetos, bem como para futuras expansões da infraestrutura e melhorias nos algoritmos implementados.

6.1 TRABALHOS FUTUROS

Embora o Cluster Aqua e o *middleware* SmS tenham demonstrado sua funcionalidade e eficiência no contexto de otimização de sistemas de engenharia, diversas oportunidades de aprimoramento foram identificadas ao longo deste trabalho. Estas melhorias buscam não apenas superar limitações observadas, mas também ampliar as capacidades do cluster e as funcionalidades do *middleware*, tornando-os mais robustos e versáteis para atender a projetos futuros.

6.1.1 Atualização do Sistema

Uma das prioridades futuras é a atualização do sistema operacional do Cluster Aqua, garantindo conformidade com as versões mais atuais dos guias e pacotes disponibilizados pela comunidade OpenHPC. Essa atualização permitirá o acesso a novos recursos, melhorias de desempenho e correções de segurança, além de ampliar a compatibilidade com ferramentas modernas de gerenciamento e escalonamento.

6.1.2 Configuração do Nodo de GPUs

O `nodo05`, atualmente subutilizado, será configurado para explorar todo o seu potencial computacional. Isso inclui a instalação de drivers específicos para suas GPUs, bem como de *softwares* e bibliotecas voltados ao desenvolvimento e execução de aplicações paralelas baseadas na Arquitetura de Dispositivo de Computação Unificada (*Compute Unified Device Architecture* – CUDA), uma API da NVIDIA para programação de GPUs. A configuração desse nodo possibilitará uma ampliação significativa das capacidades do Cluster Aqua.

6.1.3 Sistemas de Arquivos de Alto Desempenho

Outra direção é a ampliação das capacidades de armazenamento do cluster. Isso inclui a adição de dispositivos de armazenamento físico nos nodos de computação, bem como a implementação de um sistema de arquivos distribuído de alto desempenho. Soluções como Lustre ou HDFS serão avaliadas, considerando sua capacidade de lidar com altas taxas de I/O e facilitar o compartilhamento eficiente de dados entre os nodos do cluster.

6.1.4 Integração e Aplicações Futuras

Finalmente, o Cluster Aqua e o SmS podem ser integrados a outros projetos em andamento no NSO, estendendo sua aplicação a diferentes áreas de pesquisa e desenvolvimento. Projetos envolvendo aprendizado de máquina, otimização estrutural avançada e análise de grandes volumes de dados podem se beneficiar da infraestrutura aprimorada e do *middleware* desenvolvido, ampliando ainda mais o impacto deste trabalho.

REFERÊNCIAS

ADVANCED MICRO DEVICES. **Placas aceleradoras adaptáveis AMD Alveo**. 2024. Disponível em: https://www.amd.com/pt/products/accelerators/alveo.html. Acesso em: 18 nov. 2024.

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. *In*: **Proceedings** of the SPRING JOINT COMPUTER CONFERENCE, 1967. Atlantic City, p. 483–485, 1967.

ANDERSON, D.; TANNEHILL, J.; PLETCHER, R. **Computational fluid mechanics and heat transfer**. 3. ed. Boca Raton: CRC Press, 2016.

BARNEY, B.; FREDERICK, D. **Introduction to parallel computing**. Livermore, 2023. Disponível em: https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial. Acesso em: 14 nov. 2023.

BERTSEKAS, D.; TSITSIKLIS, J. **Parallel and Distributed Computation: Numerical Methods**. Belmont: Athena Scientific, 2015.

BLACK, U. **Computer networks protocols, standards, and interfaces**. Nova Jersey: Prentice-Hall, 1993.

BLAS. **Basic Linear Algebra Subprograms**. 2024. Disponível em: https://www.mpich.org/. Acesso em: 18 nov. 2024.

CENTOS. **The CentOS Project**. 2024. Disponível em: https://www.centos.org/. Acesso em: 18 nov. 2024.

CHANDRA, R. **Parallel Programming in OpenMP**. São Fransisco: Academic Press, 2001.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. **Communication of the ACM**, v. 51, n. 1, p. 107–113, jan. 2008.

DELL. **Workstations Torre e Rack Precision**. 2024. Disponível em: https://www.dell.com/pt-br/shop/isv-workstations-certificadas/sf/precision-desktops. Acesso em: 18 nov. 2024.

DOLBEAU, R. Theoretical peak flops per instruction set: a tutorial. **The Journal of Supercomputing**, v. 74, n. 3, p. 1341–1377, 2018.

DONGARRA, J. J. Performance of various computers using standard linear equations software in a fortran environment. **Simulation**, v. 49, n. 2, p. 51–62, 1987.

EIJKHOUT, V. **Introduction to high performance scientific computing**. 2. ed. Ausatin: Texas Advanced Computing Center, 2010.

FLYNN, M. Very high-speed computing systems. **IEEE**, v. 54, n. 12, p. 1901–1909, 1966.

FOSTER, I. **Designing and building parallel programs**: concepts and tools for parallel software engineering. Boston: Addison-Wesley, 2019.

GABARDO, A.; JASKOWIAK, P.; TANCREDI, T. Synapse meets slurm: Proposta de um middleware para paralelização de algoritmos de otimização populacionais. *In*: **Anais** da XXIV ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL, 2024. Florianópolis, p. 1–4, 2024.

GROPP, W.; LUSK, E.; SKJELLUM, A. **Using MPI: portable parallel programming with the message-passing interface**. Londres: MIT press, 1999. v. 1.

GUNTHER, N. J. A simple capacity model of massively parallel transaction systems. *In*: **Proceedings** of the INTERNATIONAL COMPUTER MEASUREMENT GROUP CONFERENCE, 1993. São Diego, p. 1–9, 1993.

GUNTHER, N. J. A general theory of computational scalability based on rational functions. **arXiv preprint**, 2008.

GUSTAFSON, J. L. Reevaluating amdahl's law. **Commun. ACM**, v. 31, n. 5, p. 532–533, 1988.

HAGER, G.; WELLEIN, G. **Introduction to high performance computing for scientists and engineers**. 2. ed. Boca Raton: Taylor & Francis Group, 2019.

HEWLETT-PACKARD. **PC HP Compaq 8200 Elite, fator forma pequeno**. 2024. Disponível em: https://support.hp.com/br-pt/product/details/hp-compaq-8200-elite-small-form-factor-pc/5037931. Acesso em: 18 nov. 2024.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Cambridge: MIT Press, 1992.

HPE. **HPE Cray Supercomputing Operating System Software**. 2024. Disponível em: https://www.hpe.com/psnow/doc/a00067725enw. Acesso em: 18 nov. 2024.

HPL. **A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers**. 2024. Disponível em: https://www.netlib.org/benchmark/hpl/. Acesso em: 18 nov. 2024.

JAIN, N.; BHATELE, A.; WHITE, S.; GAMBLIN, T.; KALE, L. V. Evaluating hpc networks via simulation of parallel workloads. *In*: **Proceedings** of the INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, 2016. Salt Lake City, p. 154–165, 2016.

JALOTE, P. **Fault tolerance in distributed systems**. Nova Jersey: Prentice-Hall, 1994.

KASIM, H.; MARCH, V.; ZHANG, R.; SEE, S. Survey on parallel programming model. *In*: **Proceedings** of the IFIP INTERNATIONAL CONFERENCE ON NETWORK AND PARALLEL COMPUTING, 2008. Shanghai, p. 266–275, 2008.

KINDRATENKO, V.; TRANCOSO, P. Trends in high-performance computing. **Computing in Science & Engineering**, IEEE, v. 13, n. 3, p. 92–95, 2011.

KURTZER, G. M.; JENNINGS, M. E.; MUNKI, K.; SONG, K. **Warewulf 3.0**. Berkeley, 2012.

LEDMI, A.; BENDJENNA, H.; HEMAM, S. M. Fault tolerance in distributed systems: A survey. *In*: **Proceedings** of the 3RD INTERNATIONAL CONFERENCE ON PATTERN ANALYSIS AND INTELLIGENT SYSTEMS (PAIS), 2018. Tebessa, p. 1–5, 2018.

LIANG, G. **Network Protocols**. Nova Iorque: Nova Science, 2012.

MARIADB. **MariaDB Server: the innovative open source database**. 2024. Disponível em: https://mariadb.org/. Acesso em: 18 nov. 2024.

MINYARD, C. **IPMI – a gentle introduction with OpenIPMI**. Santa Clara, 2006.

MISHRA, P.; SHUKLA, S.; ZILIC, Z. A brief history of multiprocessors and eda. **IEEE Design & Test of Computers**, v. 28, n. 03, p. 96, 2011.

MOORE, G. E. Cramming more components onto integrated circuits. **Proceedings of the IEEE**, v. 86, n. 1, p. 82–85, 1965.

MPICH. **High-Performance Portable MPI**. 2024. Disponível em: https://www.mpich.org/. Acesso em: 18 nov. 2024.

MYSQL. **The world's most popular open source database**. 2024. Disponível em: https://www.mysql.com/. Acesso em: 18 nov. 2024.

NAGEL, W. E.; KRÖNER, D. H.; RESCH, M. M. **High performance computing in science and engineering '20**: transactions of the high performance computing center, Stuttgart (HLRS) 2020. Cham: Springer, 2022.

NAGEL, W. E.; KRÖNER, D. H.; RESCH, M. M. **High performance computing in science and engineering '21**: transactions of the high performance computing center, Stuttgart (HLRS) 2021. Cham: Springer, 2023.

NAUGLE, M. G. **Network protocol handbook**. Nova Iorque: McGraw-Hill, 1992.

VON NEUMANN, J. First Draft of a Report on the EDVAC. **IEEE Annals of the History of Computing**, v. 15, n. 4, p. 27–75, 1945.

NOCEDAL, J.; WRIGHT, S. **Numerical Optimization**. Nova Iorque: Springer, 2006. (Springer Series in Operations Research and Financial Engineering).

NOERGAARD, T. **Embedded Systems Architecture**: A comprehensive guide for engineers and programmers. Oxford: Elsevier, 2012.

NVIDIA. **Jetson Modules**. 2024. Disponível em: https://developer.nvidia.com/embedded/jetson-modules. Acesso em: 18 nov. 2024.

NVIDIA. **Soluções InifiBand aceleradas para HPC**. 2024. Disponível em: https://www.nvidia.com/pt-br/networking/products/infiniband/. Acesso em: 18 nov. 2024.

OAK RIDGE NATIONAL LABORATORY. **Direction of Discovery: ORNL's exascale supercomputer is delivering world-leading performance in 2022 and beyond**. 2022. Disponível em: https://www.olcf.ornl.gov/frontier/. Acesso em: 30 nov. 2024.

OPEN MPI. **A High Performance Message Passing Library**. 2024. Disponível em: https://www.open-mpi.org/. Acesso em: 18 nov. 2024.

OPEN MULTI-PROCESSING. **The OpenMP API specification for parallel programming**. 2024. Disponível em: https://www.openmp.org/. Acesso em: 18 nov. 2024.

OPENHPC. **Open High Performance Computing Community**. 2018. Disponível em: https://openhpc.community/. Acesso em: 26 out. 2023.

PACHECO, P. **An introduction to parallel programming**. Burlington: Elsevier, 2011.

PATTERSON, D. A.; BROOKS JR, F. P.; SUTHERLAND, I. E.; THACKER, C. P. **Computer architecture**. Mariland: Elsevier Science, 2011.

PIERNAS, J.; NIEPLOCHA, J.; FELIX, E. J. Evaluation of active storage strategies for the lustre parallel file system. *In*: **Proceedings** of the 2007 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2007. Reno, p. 1–10, 2007.

POST, D.; KENDALL, R.; LUCAS, R. The opportunities, challenges, and risks of high performance computing in computational science and engineering. **Advances in Computers**, v. 66, p. 240–300, 2006.

RANKIN, K. Pxe magic: flexible network booting with menus. **Linux Journal**, v. 2008, n. 168, p. 3, 2008.

RAUBER, T.; RÜNGER, G. **Parallel Programming: for Multicore and Cluster Systems**. Cham: Springer, 2023.

RED HAT. **Red Hat Enterprise Linux Operating System**. 2024. Disponível em: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux/. Acesso em: 18 nov. 2024.

RÜDE, U.; WILLCOX, K.; MCINNES, L. C.; STERCK, H. D. Research and education in computational science and engineering. **SIAM Review**, v. 60, n. 3, p. 707–754, 2018.

RUPP, K. **50 Years of Microprocessor Trend Data**. Viena: GitHub, 2022. https://github.com/karlrupp/microprocessor-trend-data.

SANDIA NATIONAL LABORATORIES. **Sandia's ASCI Red, world's first teraflop supercomputer, is decommissioned**. 2006. Disponível em: https://newsreleases.sandia.gov/releases/2006/asci-red-decom.html. Acesso em: 30 nov. 2024.

SARANGI, S. R. **Basic Computer Architecture**. Punjab: White Falcon Publishing, 2021.

SCHAFFER, H. E. *et al.* Ncsu's virtual computing lab: A cloud computing solution. **Computer**, IEEE, v. 42, n. 7, p. 94–97, 2009.

SHVACHKO, K.; KUANG, H.; RADIA, S.; CHANSLER, R. The hadoop distributed file system. *In*: **Proceedings** of the IEEE 26TH SYMPOSIUM ON MASS STORAGE SYSTEMS AND TECHNOLOGIES (MSST), 2010. Incline Village, p. 1–10, 2010.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts**. Londres: Wiley, 2021.

SKINNER, D.; KRAMER, W. Understanding the causes of performance variability in hpc workloads. *In*: **Proceedings** of the IEEE WORKLOAD CHARACTERIZATION SYMPOSIUM, 2005., 2005. Austin, p. 137–149, 2005.

SMANEOTO, R. M.; PEREIRA, T. E.; BRASILEIRO, F. V. A cloud-based batch processing system for loosely-coupled applications. *In*: **Proceedings** of the 2021 INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING WORKSHOPS (SBAC-PADW), 2021. Belo Horizonte, p. 47–52, 2021.

SOBIESZCZANSKI-SOBIESKI, J.; MORRIS, A.; TOOREN, M. van. **Multidisciplinary Design Optimization Supported by Knowledge Based Engineering**. Londres: Wiley, 2017.

SOSINSKY, B. **Networking Bible**. Indiana: Wiley, 2009.

SOUZA, A.; REZAEI, M.; LAURE, E.; TORDSSON, J. Hybrid resource management for hpc and data intensive workloads. *In*: **Proceedings** of the 19TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2019. Larnaca, p. 399–409, 2019.

SYNAPSE ENGENHARIA MULTIDISCIPLINAR. **Documentação Synapse**. Joinville, 2023. Disponível em: https://synapse.sites.ufsc.br/tutoriais/. Acesso em: 24 nov. 2023.

TANCREDI, T. P. **Otimização multidisciplinar distribuída aplicada a projetos de engenharia**. 2009. Tese (Doutorado em Engenharia Naval e Oceanica) — Escola Politecnica, Universidade de São Paulo, São Paulo, 2009.

TANENBAUM, A. **Modern operating systems**. Nova Jersey: Pearson, 2009.

TANENBAUM, A.; FEAMSTER, N.; WETHERALL, D. **Computer Networks**. Nova Iorque: Pearson Education, 2021.

TAUBENBLATT, M. A. Optical interconnects for high-performance computing. **Journal of Lightwave Technology**, IEEE, v. 30, n. 4, p. 448–457, 2011.

THANH, T. D.; MOHAN, S.; CHOI, E.; KIM, S.; KIM, P. A taxonomy and survey on distributed file systems. *In*: **Proceedings** of the 2008 FOURTH INTERNATIONAL CONFERENCE ON NETWORKED COMPUTING AND ADVANCED INFORMATION MANAGEMENT, 2008. Gyeongju, p. 144–149, 2008.

The MPI Forum. Mpi: a message passing interface. *In*: **Proceedings** of the 1993 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 1993. Portland, p. 878–883, 1993.

TOP500. **TOP500 Supercomputers List**. Sinsheim, 2023. Disponível em: https://www.top500.org/lists/top500/. Acesso em: 26 out. 2023.

TP-LINK. **TG-3468 Adaptador de Rede Gigabit PCI Express**. 2024. Disponível em: https://www.tp-link.com/br/home-networking/adapter/tg-3468/. Acesso em: 18 nov. 2024.

TP-LINK. **TL-SG2210P Switch Smart de 10 Portas com 8 Portas Gigabit PoE+ e 2 Slots SFP**. 2024. Disponível em: https://www.tp-link.com/br/business-networking/omada-switch-smart/tl-sg2210p/. Acesso em: 18 nov. 2024.

TROBEC, R.; SLIVNIK, B.; BULIĆ, P.; ROBIČ, B. **Introduction to parallel computing**: from algorithms to programming on state-of-the-art platforms. Cham: Springer, 2018.

WAREWULF PROJECT. **Cluster Management & Provisioning**. 2024. Disponível em: https://warewulf.org/. Acesso em: 18 nov. 2024.

XCAT. **xCAT – Extreme Cloud Administration Toolkit**. 2024. Disponível em: https://www.xcat.org/. Acesso em: 18 nov. 2024.

YOO, A. B.; JETTE, M. A.; GRONDONA, M. Slurm: Simple linux utility for resource management. *In*: **Proceedings** of the WORKSHOP ON JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, 2003. Berlim, p. 44–60, 2003.

ZIMMERMANN, H. Osi reference model - the iso model of architecture for open systems interconnection. **IEEE Transactions on Communications**, v. 28, n. 4, p. 425–432, 1980.

APÊNDICE A – ARQUIVOS DE CONFIGURAÇÃO

Este apêndice apresenta os principais arquivos de configuração utilizados na instalação e operação do Cluster Aqua, conforme descrito no Capítulo 4. Os arquivos configuram serviços essenciais para o funcionamento do cluster, incluindo gerenciamento de rede, compartilhamento de arquivos, controle de acesso, gerenciamento de tarefas e provisionamento. A maioria desses arquivos está localizada no nodo mestre do cluster, salvo indicação contraria. As configurações listadas refletem o estado final do ambiente implementado e podem servir como referência para futuras implementações ou manutenções.

A.1 CONFIGURAÇÃO DHCP

/etc/dhcp/dhcpd.conf: Define as configurações do servidor DHCP, incluindo o intervalo de endereços IP alocados, tempo de concessão, interfaces de rede e parâmetros do ambiente PXE.

```
option conf-file code 209 = text;
option space isan;
option isan-encap-opts code 43 = encapsulate isan;
option isan.iqn code 203 = string;
option isan.root-path code 201 = string;
option space gpxe;
option gpxe-encap-opts code 175 = encapsulate gpxe;
option gpxe.bus-id code 177 = string;
option user-class-identifier code 77 = string;
option gpxe.no-pxedhcp code 176 = unsigned integer 8;
option tcode code 101 = text;
option iscsi-initiator-iqn code 203 = string;
ddns-update-style interim;
ignore client-updates;
option client-architecture code 93 = unsigned integer 16;
option tcode "America/Sao_Paulo";
option gpxe.no-pxedhcp 1;
option www-server code 114 = string;
option cumulus-provision-url code 239 = text;

omapi-port 7911;
key xcat_key {
  algorithm hmac-md5;
  secret "****";
};
omapi-key xcat_key;
class "pxe" {
   match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
```

```
29     ddns-updates off;
30      max-lease-time 600;
31  }
32  shared-network enp3s0 { # RedeUFSC
33    subnet 150.162.216.0 netmask 255.255.254.0 {
34      authoritative;
35      max-lease-time 43200;
36      min-lease-time 43200;
37      default-lease-time 43200;
38      option routers  150.162.217.254;
39      next-server  150.162.217.221;
40      option log-servers 150.162.217.221;
41      option ntp-servers 150.162.217.221;
42      option domain-name "lasin.br";
43      option domain-name-servers  10.0.0.63;
44      option interface-mtu 1500;
45      option domain-search  "lasin.br";
46      option cumulus-provision-url "http://150.162.217.221:80/install/
            postscripts/cumulusztp";
47      zone lasin.br. {
48         primary 10.0.0.63; key xcat_key;
49      }
50      zone 216.162.150.IN-ADDR.ARPA. {
51         primary 10.0.0.63; key xcat_key;
52      }
53      zone 217.162.150.IN-ADDR.ARPA. {
54         primary 10.0.0.63; key xcat_key;
55      }
56      if option user-class-identifier = "xNBA" and option client-architecture
            = 00:00 { #x86, xCAT Network Boot Agent
57          always-broadcast on;
58          filename = "http://150.162.217.221:80/tftpboot/xcat/xnba/nets
               /150.162.216.0_23";
59      } else if option user-class-identifier = "xNBA" and option client-
           architecture = 00:09 { #x86, xCAT Network Boot Agent
60          filename = "http://150.162.217.221:80/tftpboot/xcat/xnba/nets
               /150.162.216.0_23.uefi";
61      } else if option client-architecture = 00:00  { #x86
62          filename "xcat/xnba.kpxe";
63      } else if option vendor-class-identifier = "Etherboot-5.4"  { #x86
64          filename "xcat/xnba.kpxe";
65      } else if option client-architecture = 00:07 { #x86_64 uefi
66           filename "xcat/xnba.efi";
67      } else if option client-architecture = 00:09 { #x86_64 uefi alternative
            id
68           filename "xcat/xnba.efi";
69      } else if option client-architecture = 00:02 { #ia64
```

```
70          filename "elilo.efi";
71      } else if option client-architecture = 00:0e { #OPAL-v3
72          option conf-file = "http://150.162.217.221:80/tftpboot/pxelinux.
               cfg/p/150.162.216.0_23";
73      } else if substring (option vendor-class-identifier,0,11) = "
           onie_vendor" {
74          option www-server = "http://150.162.217.221:80/install/onie/onie-
               installer";
75      } else if substring(filename,0,1) = null {
76          filename "/yaboot";
77      }
78    } # 150.162.216.0/255.255.254.0 subnet_end
79  } # enp3s0 nic_end
80
81  shared-network eno1 { # VLAN Cluster Aqua
82    subnet 10.0.0.0 netmask 255.255.255.0 {
83      authoritative;
84      max-lease-time 43200;
85      min-lease-time 43200;
86      default-lease-time 43200;
87      option routers  10.0.0.63;
88      next-server  10.0.0.63;
89      option log-servers 10.0.0.63;
90      option ntp-servers 10.0.0.63;
91      option domain-name "lasin.br";
92      option domain-name-servers  10.0.0.63;
93      option interface-mtu 1500;
94      option domain-search  "lasin.br";
95      option cumulus-provision-url "http://10.0.0.63:80/install/postscripts/
           cumulusztp";
96      zone lasin.br. {
97         primary 10.0.0.63; key xcat_key;
98      }
99      zone 0.0.10.IN-ADDR.ARPA. {
100        primary 10.0.0.63; key xcat_key;
101     }
102     if option user-class-identifier = "xNBA" and option client-architecture
            = 00:00 { #x86, xCAT Network Boot Agent
103         always-broadcast on;
104         filename = "http://10.0.0.63:80/tftpboot/xcat/xnba/nets/10.0.0.0_24
               ";
105     } else if option user-class-identifier = "xNBA" and option client-
           architecture = 00:09 { #x86, xCAT Network Boot Agent
106         filename = "http://10.0.0.63:80/tftpboot/xcat/xnba/nets/10.0.0.0_24
               .uefi";
107     } else if option client-architecture = 00:00  { #x86
108         filename "xcat/xnba.kpxe";
```

```
109     } else if option vendor-class-identifier = "Etherboot-5.4"  { #x86
110         filename "xcat/xnba.kpxe";
111     } else if option client-architecture = 00:07 { #x86_64 uefi
112          filename "xcat/xnba.efi";
113     } else if option client-architecture = 00:09 { #x86_64 uefi alternative
            id
114          filename "xcat/xnba.efi";
115     } else if option client-architecture = 00:02 { #ia64
116          filename "elilo.efi";
117     } else if option client-architecture = 00:0e { #OPAL-v3
118          option conf-file = "http://10.0.0.63:80/tftpboot/pxelinux.cfg/p
                /10.0.0.0_24";
119     } else if substring (option vendor-class-identifier,0,11) = "
            onie_vendor" {
120          option www-server = "http://10.0.0.63:80/install/onie/onie-
                installer";
121     } else if substring(filename,0,1) = null {
122          filename "/yaboot";
123     }
124   } # 10.0.0.0/255.255.255.0 subnet_end
125 } # eno1 nic_end
126
127 host switch {
128   hardware ethernet xx:xx:xx:xx:xx:xx;
129   fixed-address 10.0.0.60;
130 }
```

/var/lib/dhcpd/dhcpd.leases: Contém a associação de endereços IP aos endereços MAC dos nodos de computação do cluster. Para brevidade algumas entradas foram omitidas.

```
1  authoring-byte-order little-endian;
2
3  host node01 {
4    dynamic;
5    hardware ethernet xx:xx:xx:xx:xx:xx;
6    uid xx:xx:xx:xx:xx:xx;
7    fixed-address xx.x.x.x;
8          supersede server.ddns-hostname = "node01";
9          supersede host-name = "node01";
10         if option user-class-identifier = "xNBA" and option client-
               architecture
11               = 00:00 {
12           supersede server.always-broadcast = 01;
13           supersede server.filename =
14                   "http://${next-server}:80/tftpboot/xcat/xnba/nodes/node01
                        ";
15         } elsif option user-class-identifier = "xNBA" and option
```

```
16                 client-architecture = 00:09 {
17           supersede server.filename =
18                                     "http://${next-server}:80/tftpboot/
                                          xcat/xnba/nodes/node01.uefi";
19         } elsif option client-architecture = 00:07 {
20           supersede server.filename = "xcat/xnba.efi";
21         } elsif option client-architecture = 00:00 {
22           supersede server.filename = "xcat/xnba.kpxe";
23         } else {
24           supersede server.filename = "";
25         }
26 }
27
28 host node02 {
29   dynamic;
30   hardware ethernet xx:xx:xx:xx:xx:xx;
31   uid xx:xx:xx:xx:xx:xx;
32   fixed-address xx.x.x.x;
33         supersede server.ddns-hostname = "node02";
34         supersede host-name = "node02";
35         if option user-class-identifier = "xNBA" and option client-
              architecture
36               = 00:00 {
37           supersede server.always-broadcast = 01;
38           supersede server.filename =
39                   "http://${next-server}:80/tftpboot/xcat/xnba/nodes/node02
                        ";
40         } elsif option user-class-identifier = "xNBA" and option
41                 client-architecture = 00:09 {
42           supersede server.filename =
43                                     "http://${next-server}:80/tftpboot/
                                          xcat/xnba/nodes/node02.uefi";
44         } elsif option client-architecture = 00:07 {
45           supersede server.filename = "xcat/xnba.efi";
46         } elsif option client-architecture = 00:00 {
47           supersede server.filename = "xcat/xnba.kpxe";
48         } else {
49           supersede server.filename = "";
50         }
51 }
52
53 ...
```

A.2 CONFIGURAÇÃO NFS

/etc/exports: Especifica os diretórios do nodo mestre que são compartilhados via NFS e suas permissões de acesso.

```
/home            *(rw,no_subtree_check,fsid=10,no_root_squash)
/opt/ohpc/pub    *(ro,no_subtree_check,fsid=11)
/opt/intel       *(ro,no_subtree_check,fsid=12)
/opt/openfoam    *(rw,no_subtree_check,fsid=13,no_root_squash)
/opt/ansys       *(rw,no_subtree_check,fsid=14,no_root_squash)
```

/etc/fstab: arquivo localizado nos nodos de computação, utilizado para a montagem automática dos diretórios compartilhados pelo nodo mestre via NFS.

```
proc                    /proc           proc    rw 0 0
sysfs                   /sys            sysfs   rw 0 0
devpts                  /dev/pts        devpts  rw,gid=5,mode=620 0 0
10.0.0.63:/home         /home           nfs     nfsvers=3,nodev,nosuid 0 0
10.0.0.63:/opt/ohpc/pub /opt/ohpc/pub   nfs     nfsvers=3,nodev 0 0
10.0.0.63:/opt/intel    /opt/intel      nfs     nfsvers=3,nodev 0 0
10.0.0.63:/opt/openfoam /opt/openfoam   nfs     nfsvers=3,nodev 0 0
10.0.0.63:/opt/ansys    /opt/ansys      nfs     nfsvers=3,nodev 0 0
```

A.3 CONFIGURAÇÃO IPTABLES

/etc/sysconfig/iptables: Define as regras de *firewall* do nodo mestre, garantindo segurança na comunicação entre nodos e com a rede externa. Permite compartilhamento de dados apenas em portas estritamente necessárias.

```
1  *filter
2  -F
3  -X
4
5  # politica padrao do firewall eh dropar todos os pedidos
6  -P INPUT DROP
7  -P FORWARD DROP
8  -P OUTPUT DROP
9
10 # trafego ilimitado na rede interna do cluster
11 -A INPUT -i eno1 -j ACCEPT
12 -A FORWARD -i eno1 -j ACCEPT
13 -A OUTPUT -o eno1 -j ACCEPT
14
15 # trafego ilimitado em loopback (localhost)
16 -A INPUT -i lo -j ACCEPT
17 -A FORWARD -i lo -j ACCEPT
18 -A OUTPUT -o lo -j ACCEPT
19
20 ############################################################
21 ##              Regras especifas por servico                ##
22 ############################################################
23
24 ## ssh
25 -A INPUT -p tcp --dport xxx -j ACCEPT
26 -A FORWARD -p tcp --dport xxx -j ACCEPT
27 -A OUTPUT -p tcp --sport xxx -j ACCEPT
28
29 ## portmap/rpcbind
30 -A INPUT -p tcp ! -s 10.0.0.0/32 --dport 111 -j DROP
31 -A INPUT -p tcp -s 127.0.0.1 --dport 111 -j ACCEPT
32 -A INPUT -p udp ! -s 10.0.0.0/32 --dport 111 -j DROP
33
34 ## HTTP
35 -A INPUT -p tcp -m tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j
       ACCEPT
36 -A OUTPUT -p tcp -m tcp --sport 80 -m conntrack --ctstate ESTABLISHED -j
       ACCEPT
37 -A INPUT -p tcp -m tcp --dport 8000 -m conntrack --ctstate NEW,ESTABLISHED
       -j ACCEPT
38 -A OUTPUT -p tcp -m tcp --sport 8000 -m conntrack --ctstate ESTABLISHED -j
       ACCEPT
```

```
39
40 ## OpenSSL
41 -A INPUT -p tcp -m tcp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -
       j ACCEPT
42 -A OUTPUT -p tcp -m tcp --sport 443 -m conntrack --ctstate ESTABLISHED -j
       ACCEPT
43
44 ## Cockpit
45 -A INPUT -p tcp -m tcp --dport 9090 -m conntrack --ctstate NEW,ESTABLISHED
       -j ACCEPT
46 -A OUTPUT -p tcp -m tcp --sport 9090 -m conntrack --ctstate ESTABLISHED -j
       ACCEPT
47
48 ## DNS & OHPC
49 -A OUTPUT -p udp --dport 53 -j ACCEPT
50 -A INPUT -i enp3s0 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
51
52 ## OHPC
53 -A OUTPUT -d 18.190.130.150 -p tcp --dport 80 -j ACCEPT
54
55 ## XCAT
56 -A OUTPUT -d 166.70.135.166 -p tcp --dport 80 -j ACCEPT
57
58 ## Intel oneAPI
59 -A OUTPUT -d 23.50.77.206 -p tcp --dport 80 -j ACCEPT
60
61 ## OpenFOAM
62 -A OUTPUT -d 82.71.205.33 -p tcp --dport 80 -j ACCEPT
63
64 ## OpenFOAM
65 -A INPUT -p tcp -s 150.162.216.59 -j ACCEPT
66 -A OUTPUT -p tcp -d 150.162.216.59 -j ACCEPT
67
68 ## R CRAN repos
69 # CRAN US
70 -A OUTPUT -d 137.208.57.46 -p tcp --dport 80 -j ACCEPT
71 # CRAN UFPR
72 -A OUTPUT -d 200.236.31.1 -p tcp --dport 80 -j ACCEPT
73 # CRAN FIOCRUZ
74 -A OUTPUT -d 157.86.96.98 -p tcp --dport 80 -j ACCEPT
75
76 #Ansys RSM
77 -A INPUT -p tcp --dport 9241 -j ACCEPT
78 -A INPUT -p tcp --dport 1000:3000 -j ACCEPT
79 -A OUTPUT -p tcp --sport 1000:3000 -j ACCEPT
80 -A OUTPUT -p tcp --sport 9241 -j ACCEPT
81
```

```
-A FORWARD -i eno1 -o enp3s0 -j ACCEPT
-A FORWARD -i enp3s0 -o eno1 -m state --state RELATED,ESTABLISHED -j ACCEPT

############################################################
# redundancia do bloqueio de portas de acesso publico

-A INPUT -i enp3s0 -j DROP
-A FORWARD -i enp3s0 -j DROP
-A OUTPUT -o enp3s0 -j DROP

COMMIT
```

A.4 CONFIGURAÇÃO SSH

/etc/ssh/sshd_config: Configurações do servidor SSH, permite o acesso remoto seguro ao cluster e mudança porta padrão de conexão.

```
1  Port XXXX
2
3  HostKey /etc/ssh/ssh_host_rsa_key
4  HostKey /etc/ssh/ssh_host_ecdsa_key
5  HostKey /etc/ssh/ssh_host_ed25519_key
6
7  # Logging
8  SyslogFacility AUTHPRIV
9
10 # Authentication:
11 PermitRootLogin no
12 AuthorizedKeysFile      .ssh/authorized_keys
13 PasswordAuthentication yes
14 ChallengeResponseAuthentication no
15
16 # GSSAPI options
17 GSSAPIAuthentication yes
18 GSSAPICleanupCredentials no
19
20 UsePAM yes
21 X11Forwarding yes
22 PrintMotd no
23
24 # Accept locale-related environment variables
25 AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY
      LC_MESSAGES
26 AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
27 AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
28 AcceptEnv XMODIFIERS
29
30 Subsystem       sftp    /usr/libexec/openssh/sftp-server
```

A.5 CONFIGURAÇÃO XCAT

/tftpboot/xcat/xnba/nodes/<nodename>.uefi: Configuração da inicialização via PXE dos nodos de computação, especificando as imagens e parâmetros necessários.

```
1 #!gpxe
2 imgfetch -n kernel http://${next-server}:80/tftpboot/xcat/osimage/centos-
    stream8-x86_64-netboot-compute/kernel
3 imgload kernel
4 imgargs kernel imgurl=http://${next-server}:80//install/netboot/centos-
    stream8/x86_64/compute/rootimg.cpio.gz XCAT=${next-server}:3001 NODE=
    node01 FC=0 XCATHTTPPORT=80  console=tty0 console=ttyS0,115200 BOOTIF
    =01-${netX/mac:hexhyp} initrd=initrd
5 imgfetch -n initrd http://${next-server}:80/tftpboot/xcat/osimage/centos-
    stream8-x86_64-netboot-compute/initrd-stateless.gz
6 imgexec kernel
```

A.6 CONFIGURAÇÃO SLURM

/etc/slurm/slurm.conf: Define os parâmetros do escalonador de tarefas, como nodos disponíveis, filas e políticas de agendamento.

```
ClusterName=aqua
SlurmctldHost=xxx.xxx.xxx.xx

MpiDefault=none
ProctrackType=proctrack/linuxproc
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmctldPort=xxx
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=xxx
SlurmdSpoolDir=/var/spool/slurmd
SlurmUser=slurm
StateSaveLocation=/var/spool/slurmctld
SwitchType=switch/none

# TIMERS
InactiveLimit=0
KillWait=30
MinJobAge=300
SlurmctldTimeout=120
SlurmdTimeout=300
Waittime=0

# SCHEDULING
SchedulerType=sched/backfill
SelectType=select/cons_tres
SelectTypeParameters=CR_Core

# LOGGING AND ACCOUNTING
AccountingStorageEnforce=limits
AccountingStorageHost=localhost
AccountingStoragePass=/var/run/munge/munge.socket.2
AccountingStoragePort=7031
AccountingStorageType=accounting_storage/slurmdbd
AccountingStorageUser=slurm
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/linux
SlurmctldDebug=info
SlurmctldLogFile=/var/log/slurmctld.log
SlurmdDebug=info
SlurmdLogFile=/var/log/slurmd.log

# COMPUTE NODES
TaskPlugin=task/affinity
```

```
PropagateResourceLimitsExcept=MEMLOCK
JobCompType=jobcomp/filetxt
Epilog=/etc/slurm/slurm.epilog.clean
NodeName=node01 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN
NodeName=node02 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN
NodeName=node03 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN
NodeName=node04 Sockets=2 CoresPerSocket=12 ThreadsPerCore=2 State=UNKNOWN
NodeName=node05 Sockets=2 CoresPerSocket=6 ThreadsPerCore=1 Gres=gpu:4
    State=UNKNOWN
NodeName=node06 Sockets=1 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN
NodeName=node07 Sockets=1 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
PartitionName=normal Nodes=node0[1-4] Default=YES MaxTime=72:00:00 State=UP
    Oversubscribe=YES
PartitionName=gpu Nodes=node05 Default=NO MaxTime=INFINITE State=UP
    Oversubscribe=YES
PartitionName=lowmem Nodes=node0[5-7] Default=NO MaxTime=24:00:00 State=UP
    Oversubscribe=YES
ReturnToService=1
HealthCheckProgram=/usr/sbin/nhc
HealthCheckInterval=300
```

/etc/slurm/slurmdb.conf: Configura a conexão com o banco de dados do SLURM, armazenando informações sobre as tarefas e o uso de recursos no cluster.

```
# Archive info
ArchiveJobs=yes
ArchiveDir="/tmp"
ArchiveSteps=yes

# Authentication info
AuthType=auth/munge

# slurmDBD info
DbdAddr=xxx.xxx.xxx.xx
DbdHost=xxx.xxx.xxx.xx
DbdPort=XXXX
SlurmUser=slurm
DebugLevel=verbose
LogFile=/var/log/slurm/slurmdbd.log
PidFile=/var/run/slurmdbd.pid

# Database info
StorageType=accounting_storage/mysql
StorageHost=localhost
StoragePass=/var/run/munge/munge.socket.2
StorageUser=slurm
StorageLoc=slurm_acct_db
```