Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Programa de Pós-Graduação em Ciência da Computação

**Alexandre Tavares de Oliveira**

# Uma Plataforma de Rede Definida por Software para Ambientes de Computação Paralela e Distribuída

Juiz de Fora

2019

**Alexandre Tavares de Oliveira**

# Uma Plataforma de Rede Definida por Software para Ambientes de Computação Paralela e Distribuída

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Alex Borges Vieira

Coorientadores: Antônio Tadeu Azevedo Gomes, Artur Ziviani

Juiz de Fora

2019

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Oliveira, Alexandre Tavares de.
Uma Plataforma de Rede Definida por Software para Ambientes de Computação Paralela e Distribuída / Alexandre Tavares de Oliveira. – 2019.
65 f. : il.

Orientador: Alex Borges Vieira
Coorientadores: Antônio Tadeu Azevedo Gomes, Artur Ziviani
Dissertação (Mestrado Acadêmico) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2019.

1. Redes definidas por software. 2. Computação paralela. 3. Computação distribuída. I. Vieira, Alex Borges, orient. II. Gomes, Antônio Tadeu Azevedo, coorient. III. Ziviani, Artur, coorient. IV. Título.

**Alexandre Tavares de Oliveira**

**Uma Plataforma de Rede Definida por Software para Ambientes de Computação Paralela e Distribuída**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: 26 de fevereiro de 2019.

BANCA EXAMINADORA

---

Professor Dr. Alex Borges Vieira - Orientador
Universidade Federal de Juiz de Fora

---

Professor Dr. Antônio Tadeu Azevedo Gomes - Coorientador
Laboratório Nacional de Computação Científica

---

Professor Dr. Artur Ziviani - Coorientador
Laboratório Nacional de Computação Científica

---

Professor Dr. Luiz Henrique Andrade Correia
Universidade Federal de Lavras

---

Professor Dr. Mario Antonio Ribeiro Dantas
Universidade Federal de Juiz de Fora

*À minha esposa Josiane e à minha filha Lívia, pelo amor e apoio incondicionais, e por suportarem os vários momentos de ausência, tanto física quanto espiritual.*

*Aos meus pais, Gilberto e Sueli, pelos exemplos de caráter, dedicação e perseverança.*

## AGRADECIMENTOS

A Deus, por guiar os meus passos, iluminar o meu caminho e conduzir minhas ações durante esta jornada.

À toda minha família, pelas várias manifestações de apoio ao longo de todo este processo. Um agradecimento especial à minha irmã Valéria, minha referência de inteligência e que sempre esteve pronta a revisar minhas traduções.

À Polícia Rodoviária Federal, por oferecer todas as condições de me dedicar integralmente ao curso, em especial aos Inspetores De Lima, Fabio Williams e José Carlos, por reconhecerem meu valor e a importância dessa formação para o meu crescimento pessoal e profissional. Obrigado ainda a todos os colegas do Nutic/RJ, por assumirem minhas responsabilidades durante o afastamento.

Ao meu orientador, Prof. Dr. Alex Borges Vieira, por confiar em mim e no meu trabalho, dando-me a liberdade necessária para o desenvolvimento das atividades. Agradeço imensamente a paciência, a dedicação e o comprometimento. Seus ensinamentos foram fundamentais para o meu sucesso; os levarei comigo pra sempre.

Aos meus coorientadores, Prof. Dr. Antônio Tadeu de Azevedo Gomes e Prof. Dr. Artur Ziviani, por abdicarem de seus preciosos tempos em favor de um velho amigo e aluno. Obrigado pelas recomendações e por todas as contribuições a este trabalho. Agradeço também por me acolherem nas dependências do LNCC.

Aos professores e funcionários do Programa de Pós-Graduação em Ciência da Computação da UFJF, pelo empenho e esforço, tanto na difusão do conhecimento quanto no suporte às atividades do curso.

A todos os colegas do curso, veteranos, companheiros de turma e contemporâneos, pelas dicas e pelos momentos de descontração. Agradeço a todos os integrantes do NetLab pelo ambiente agradável, acolhedor e, acima de tudo, enriquecedor.

Aos servidores, colaboradores e alunos do LNCC, principalmente aos integrantes do laboratório MARTIN. Um agradecimento especial a Iuri Malinoski, pelo apoio e prontidão no provimento de acesso a recursos computacionais.

"Aplica o teu coração à instrução e os teus
ouvidos às palavras do conhecimento."
(Provérbios 23:12)

**RESUMO**

O crescimento no volume e na diversidade dos dados causado pelo fenômeno *Big Data* tem revolucionado os negócios e a ciência, ao mesmo tempo que requer capacidade cada vez maior dos recursos computacionais. As plataformas de computação de alto desempenho (HPC), tradicionalmente empregadas em simulações numéricas massivamente paralelas, oferecem capacidade computacional que pode ser aproveitada na análise de *Big Data*. No entanto, a confluência de *Big Data* e HPC, embora pareça ser natural, deve ser examinada sob diversos aspectos, o que envolve a adequação de vários de seus elementos. Em particular, a infraestrutura de rede precisa ser eficiente e flexível para ajustar-se às demandas bem distintas das aplicações típicas desses ambientes de computação paralela e distribuída. O paradigma de rede definida por software (SDN) pode favorecer essa integração, graças à sua visão global e seu maior nível de programabilidade, que simplificam a gerência da rede e a tornam mais adaptável e efetiva. Nesse contexto, este trabalho apresenta uma plataforma SDN capaz de suprir os requisitos de desempenho de rede de aplicações *Big Data* e HPC. A plataforma busca otimizar a comunicação dos dados, identificando o tráfego de rede por meio de uma API e aplicando dinamicamente mecanismos de roteamento mais adequados a cada perfil de tráfego. Essa abordagem evidencia um modelo de rede ciente da aplicação que permite a diminuição no tempo de execução de aplicações. Avaliações mediante simulações em cenários específicos demonstram a viabilidade e a aplicabilidade da plataforma, ao reduzir o tempo médio de execução de aplicações reais MPI em cerca de 11%, e Hadoop em torno de 6%.

Palavras-chave: Redes definidas por software. Computação paralela. Computação distribuída.

**ABSTRACT**

The growth in the volume and diversity of data caused by the Big Data phenomenon has revolutionized business and science, at the same time as it demands ever-increasing computational resources. High-performance computing (HPC) platforms tailored to massively parallel numerical simulations offer computational capacity that can be leveraged by Big Data Analytics solutions. Nevertheless, the confluence of Big Data and HPC, although it appears to be natural, should be examined in several aspects, which involves the suitability of several of its elements. In particular, the network infrastructure needs to be efficient and flexible to fit the very distinct demands of typical applications of such parallel and distributed computing environments. The software-defined network paradigm (SDN) may favor this integration, thanks to its global view and its higher level of programmability, which simplify network management and make it more adaptive and effective. In this context, this work presents an SDN platform capable of supplying the network performance requirements of Big Data and HPC applications. The platform seeks to optimize data communication by identifying network traffic through an API and dynamically applying the most appropriate routing mechanisms to each traffic profile. This approach evidences an application-aware network model that allows the decrease in the execution time of applications. Evaluations through simulations in specific scenarios demonstrate the feasibility and applicability of the platform, by reducing the average execution time of MPI applications by about 11%, and of Hadoop applications by around 6%.

Keywords: Software-defined network. Parallel computing. Distributed computing.

## LISTA DE ILUSTRAÇÕES

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| ECMP | *Equal-Cost MultiPath* |
| GSMP | *General Switch Management Protocol* |
| HDFS | *Hadoop Distributed File System* |
| HPC | *High-Performance Computing* |
| HPCC | HPC *Challenge Benchmark* |
| IETF | *Internet Engineering Task Force* |
| IoT | *Internet of Things* |
| MIMD | *Multiple Instruction, Multiple Data* |
| MPI | *Message Passing Interface* |
| MPLS | *Multiprotocol Label Switching* |
| MPP | *Massively Parallel Processors* |
| NUMA | *Non-Uniform Memory Access* |
| ONF | *Open Networking Foundation* |
| POD | *Point of Delivery* |
| QoS | *Quality of Service* |
| RDMA | *Remote Direct Memory Access* |
| SDN | *Software-Defined Network* |
| SMP | *Symmetric Multiprocessors* |

**SUMÁRIO**

# 1 INTRODUÇÃO

A revolução *Big Data* é hoje amplamente reconhecida como a fase inicial de um processo de transformação da sociedade moderna (ASCH et al., 2018). Desde os anos 2010, corporações e instituições de pesquisa vêm se beneficiando amplamente de soluções originadas por esse fenômeno. Grandes empresas da Internet como Amazon, Google e Facebook aplicam metodologias de processamento de grandes volumes de dados para exame de manifestações em redes sociais, registros de operações e movimentações financeiras. A partir da análise de dados, é possível extrair informações extremamente valiosas para os negócios dessas empresas, tais como padrões e tendências de comportamento e consumo. No universo científico, aplicações de *Big Data Analytics* em meteorologia, astronomia e biologia estão revolucionando as pesquisas nessas e em muitas outras áreas do conhecimento. Até mesmo o setor público vem usufruindo das abordagens de *Big Data/Data Analytics*, otimizando sua gestão e direcionando melhor seus investimentos em setores de saúde, segurança e infraestrutura, por exemplo.

De fato, a expansão da computação móvel, a disseminação das redes sociais e o surgimento da Internet das Coisas (*Internet of Things* – IoT) vêm contribuindo para um crescimento sem precedentes no volume e na diversidade dos dados. Obviamente, o processamento de dados em larga escala impõe enormes desafios, tanto em termos de gerenciamento de memória quanto de processamento e acesso aos dados. Nesse sentido, as plataformas de computação de alto desempenho (*High-Performance Computing* – HPC) —tradicionalmente empregadas em simulações numéricas massivamente paralelas, como previsão de tempo e clima, modelagem de reservatórios de hidrocarbonetos e simulação de atracamento molecular— oferecem capacidade de processamento computacional que pode ser aproveitada por uma vasta gama de soluções de computação intensiva de dados. Embora a utilização dessas plataformas de HPC por aplicações de *Big Data* pareça ser natural, o que se vê, de fato, é um descasamento entre essas arquiteturas, de modo que explorar recursos de HPC para atender soluções típicas de *Big Data* demanda o tratamento de diversos problemas.

## 1.1 DEFINIÇÃO DO PROBLEMA

A confluência entre os atuais ambientes de computação paralela e distribuída, isto é, *frameworks* de *Big Data* e plataformas de HPC, precisa ser considerada sob diferentes aspectos, desde a tecnologia de *hardware*, *software* e aplicações/algoritmos até modelos de negócio e educação (FOX et al., 2015). A rede de dados, em particular, precisa ser eficiente e flexível para se ajustar aos diferentes perfis de tráfego gerados pelas aplicações que operam sobre a rede. Portanto, nesse cenário, uma plataforma de comunicação efetiva pressupõe o atendimento das exigências de rede de aplicações paralelas e distribuídas que

utilizam as APIs (*Application Programming Interface*) tradicionais de ambientes de HPC, como *Message Passing Interface* (MPI), e das aplicações que empregam *frameworks* típicos de ambientes *Big Data*, como Hadoop e Spark.

No entanto, as redes de dados tradicionais possuem controles complexos, o que as tornam "ossificadas". Nessas arquiteturas convencionais, a decisão sobre como manipular pacotes (plano de controle) e transferir pacotes (plano de dados) são implementadas como recursos unificados e inseparáveis em cada ativo de rede (comutadores e roteadores). Logo, o ajuste das redes de comunicação aos requisitos específicos de cada uma dessas aplicações demanda um alto custo administrativo, o que inviabiliza, na prática, o uso dessas infraestruturas de forma integrada e eficiente. Assim, o problema a ser tratado nesta dissertação é a confluência efetiva dos ambientes de computação paralela e distribuída no âmbito das redes de comunicação. Nesse aspecto, há novos paradigmas de redes que podem favorecer essa integração, como as Redes Definidas por *Software*.

## 1.2 OBJETIVO DA DISSERTAÇÃO

Redes Definidas por *Software* (*Software-Defined Networks* – SDN) constituem um paradigma emergente que facilita a criação e a introdução de novas abstrações na rede, simplificando o gerenciamento e facilitando a sua evolução (KREUTZ et al., 2015). Uma das características fundamentais do conceito SDN é o desacoplamento dos planos de dados e de controle dos ativos de rede. Com isso, a inteligência da rede é transferida para um elemento controlador externo logicamente centralizado que, através de tecnologias de comunicação seguras como o OpenFlow (MCKEOWN et al., 2008), gerencia os dispositivos de encaminhamento de dados. Dessa forma, o modelo SDN proporciona uma visão global e um maior nível de programabilidade da rede de dados, o que aumenta a flexibilidade e, consequentemente, a eficiência dessas redes, o que o torna bastante adequado no contexto dos ambientes de computação paralela e distribuída.

Considerando o contexto descrito e a relevância do problema apontado, este trabalho pretende responder à seguinte questão de pesquisa: **"como o paradigma SDN pode fornecer uma infraestrutura de rede de dados flexível capaz de suprir os requisitos de aplicações de computação paralela e distribuída?"** Assim, a partir da revisão sistemática do tema e do estudo na área, esta dissertação possui o objetivo fundamental de propor um sistema SDN para ambientes de *Big Data* e HPC.

Para cumprir esse propósito, apresentamos uma plataforma de comunicação de dados baseada em SDN que fornece uma infraestrutura de rede flexível e dinâmica, capaz de suprir os requisitos de desempenho das aplicações que executam em ambientes de *Big Data* e HPC. A plataforma procura otimizar a comunicação dos dados em redes multicaminhos mediante o emprego dos algoritmos de roteamento mais adequados aos perfis de tráfego de cada uma dessas aplicações. Para tanto, idealizamos uma API que

permite às aplicações paralelas e distribuídas informarem ao controlador SDN, em tempo de execução, características gerais dos perfis de tráfego por elas gerados, tornando a rede ciente da aplicação. Da mesma forma, a API concebida possibilita que o administrador do ambiente configure os algoritmos de roteamento mais apropriados a cada um desses perfis de tráfego. Desse modo, o controlador torna-se apto a identificar o tráfego da aplicação através de configurações reativas e/ou proativas, e a aplicar a estratégia de roteamento previamente definida a esse tráfego.

## 1.3 CONTRIBUIÇÕES DA DISSERTAÇÃO

No domínio dos ambientes de computação avançada, esta dissertação apresenta duas contribuições principais:

1. **Otimização de aplicações paralelas e distribuídas no âmbito de um ambiente integrado, com enfoque na rede de comunicação.** As propostas de soluções SDN na área de computação paralela e distribuída geralmente tratam essas classes de aplicações de forma isolada ou conscientes da rede, em oposição à nossa proposta de rede integrada e ciente da aplicação, o que dá ao administrador maior controle. Trabalhos anteriores também se referem ao tema da confluência focando outros aspectos, que não o de rede. Nesse contexto, a plataforma de comunicação proposta contribui para assegurar a flexibilidade necessária aos ajustes da infraestrutura, simplificando a gerência da rede de dados e tornando-a verdadeiramente integrada e mais eficiente. Consequentemente, as aplicações paralelas e distribuídas tendem a apresentar, de forma geral, um melhor desempenho.

2. **Discussão sobre a implantação de infraestruturas SDN Ethernet para atender soluções de HPC.** Nossas análises mostram a aplicabilidade do paradigma SDN em infraestruturas Ethernet como solução para maximizar a comunicação em redes de alto desempenho, mitigando problemas comuns dessa tecnologia como variação estatística do retardo e baixa capacidade efetiva. Nesse aspecto, este trabalho contribui para que se abra uma nova perspectiva no estudo de soluções integradas, no sentido de se ponderar sobre a implantação de infraestruturas SDN Ethernet, significativamente mais baratas, para atender soluções de HPC. Não obstante, infraestruturas de rede baseadas em tecnologias como InfiniBand ainda podem se beneficiar das capacidades do modelo SDN, tendo em vista que um dos principais fabricantes de dispositivos de redes que implementam essa tecnologia começam a fornecer produtos com suporte a SDN (MELLANOX, 2015, 2019).

Avaliamos a proposta de plataforma por meio de simulações de um ambiente SDN Ethernet, com um cenário específico de topologia multicaminhos, e apresentamos os

resultados obtidos. Nessa circunstância, propomos quatro algoritmos simples de seleção de rotas para ambientes de rede de múltiplos caminhos. Considerando esse escopo, em uma primeira avaliação, analisamos o desempenho da rede perante o tráfego HPC gerado por um *benchmark* MPI. A partir de diferentes medições, constatamos diferenças de vazão de até 39% na comparação entre as estratégias de roteamento propostas, ao mesmo tempo que a latência dos pacotes permaneceu estável. Em uma segunda análise, utilizamos um emulador de aplicações Hadoop para identificar as configurações de rede mais adequadas às aplicações típicas de ambientes *Big Data*. Medimos o tempo de conclusão da emulação também em função dos quatro algoritmos propostos. Os resultados dessa simulação mostram diferenças de até 22% nos tempos de execução, dependendo do algoritmo usado.

Finalmente, analisamos o desempenho de aplicações reais *Big Data* e HPC co-localizadas no mesmo ambiente, utilizando os mecanismos de identificação de tráfego baseados na API proposta neste trabalho. Nessa simulação, aplicamos as estratégias de roteamento mais adequadas a cada aplicação, considerando as avaliações anteriores para MPI e para Hadoop. Nessa condição, verificamos que a plataforma SDN reduziu os tempos de execução em mais de 11% para as aplicações MPI e em torno de 6% para as aplicações Hadoop, em comparação com um cenário onde é aplicado um algoritmo de roteamento genérico para ambas as aplicações.

## 1.4 CONTRIBUIÇÕES ACADÊMICAS

Os primeiros avanços e as contribuições parciais desta dissertação foram anunciados à comunidade acadêmica sob a forma de um artigo, publicado e apresentado em outubro de 2018 no XIX Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD 2018), realizado na cidade de São Paulo/SP. O trabalho, intitulado **"Análise de Desempenho de Rede para Aplicações MPI em Infraestruturas SDNs Convergentes para HPC e Big Data"** (OLIVEIRA et al., 2018c), apresenta a concepção da plataforma SDN e os resultados da avaliação de rede com tráfego HPC.

Em um contexto mais abrangente, a pesquisa na área de SDN também produziu contribuições acadêmicas relevantes que foram publicadas em anais de importantes conferências, uma nacional e outra internacional. Essas contribuições referem-se, especificamente, a um estudo sobre provisão de qualidade de serviço (QoS – *Quality of Service*) em SDN. O artigo **"Arquitetura de Provisão de Qualidade de Serviço para Aplicações Distribuídas de Alto Desempenho em Redes Definidas por Software"** (OLIVEIRA et al., 2018a) ganhou o prêmio de menção honrosa como um dos melhores trabalhos apresentados no XXIII *Workshop* de Gerência e Operação de Redes e Serviços (WGRS-SBRC 2018), realizado em maio de 2018 em Campos do Jordão/SP.

Uma versão aprimorada do estudo de QoS em SDN foi publicada e apresentada no *IEEE Symposium on Computers and Communications* (ISCC 2018) através do artigo

**"SDN-Based Architecture for Providing QoS to High Performance Distributed Applications"** (OLIVEIRA et al., 2018b). Essa conferência internacional foi realizada em junho de 2018 no Brasil, na cidade de Natal/RN.

## 1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

O restante desta dissertação está estruturado da seguinte forma. O Capítulo 2 apresenta os aspectos fundamentais do paradigma de Redes Definidas por *Software* (SDN), destacando seu desenvolvimento, definições, arquitetura e principais componentes e tecnologias. No Capítulo 3, mostramos os conceitos por trás dos ambientes de computação avançada, incluindo suas arquiteturas paralelas, redes de comunicação e modelos de programação paralela. O Capítulo 4 explora a proposta de plataforma de comunicação baseada em SDN, analisando o contexto envolvido e detalhando, entre outros, as funcionalidades, mecanismos e componentes da solução. A descrição completa das três avaliações executadas sobre a plataforma proposta é apresentada no Capítulo 5. Além de abordar o cenário considerado e a metodologia geral de análise, o Capítulo 5 detalha as metodologias específicas e os resultados obtidos na avaliação de rede com tráfego HPC, na avaliação de rede com tráfego *Big Data*, e na avaliação de desempenho de aplicações reais. Por sua vez, o Capítulo 6 discute os trabalhos relacionados. Por fim, o Capítulo 7 mostra as conclusões da dissertação e os trabalhos futuros.

# 2 REDES DEFINIDAS POR SOFTWARE

Este capítulo apresenta os aspectos fundamentais do paradigma de Redes Definidas por *Software* (SDN). Inicialmente, a Seção 2.1 descreve as motivações que estimularam as pesquisas nessa área. Na Seção 2.2 são discutidas, sob uma perspectiva histórica breve, as iniciativas que levaram ao surgimento do paradigma. Por sua vez, a Seção 2.3 apresenta algumas definições para SDN. Na Seção 2.4, exploramos a arquitetura e os componentes do modelo. Finalmente, a Seção 2.5 mostra as principais características da tecnologia OpenFlow.

## 2.1 MOTIVAÇÃO

Conceitualmente, dispositivos de redes como comutadores (*switches*) e roteadores possuem dois tipos de mecanismos: os que lidam com o processamento e encaminhamento de pacotes; e os que permitem aos operadores de rede controlarem e configurarem o dispositivo. Para essa divisão, associamos os termos *plano de dados* e *plano de controle*, respectivamente. Seguindo essa concepção, os fornecedores de equipamentos de redes geralmente seguem uma abordagem altamente integrada, na qual os planos de dados e de controle estão fortemente acoplados. Nesse caso, o plano de controle do dispositivo fornece uma interface de gerenciamento através da qual são configuradas suas funções específicas (COMER, 2015).

As redes tradicionais operam com dispositivos que seguem a lógica convencional, isto é, seus planos de dados e de controle são combinados dentro de cada nó da rede. Nessa abordagem, a política de encaminhamento de pacotes de dados é definida através da configuração especializada e individual de cada dispositivo. Portanto, o ajuste das infraestruturas de redes de comunicação demanda controles complexos e um alto custo administrativo, o que acaba restringindo a extensibilidade, a escalabilidade e a própria capacidade de configuração das redes tradicionais. Com o aumento da complexidade das redes ao longo dos anos, essas restrições acabaram gerando um gargalo importante. As redes tornaram-se "ossificadas", ou seja, sem espaço para experimentação e inovação, o que culminou com uma certa estagnação na criação de novos protocolos e serviços (COSTA et al., 2016, 2017).

Para tratar o problema de ossificação de redes, diversos estudos e pesquisas foram realizados e novas abordagens e tecnologias foram propostas, o que serviu de base para o surgimento e a evolução do conceito de Redes Definidas por *Software* (*Software-Defined Networks* – SDN). Dentre essas iniciativas, algumas delas exerceram uma grande influência no desenvolvimento do atual paradigma. A compreensão desse histórico de projetos permite um entendimento melhor da tecnologia SDN.

## 2.2 HISTÓRICO RESUMIDO DE SDN

Entre as principais iniciativas que formaram o alicerce sobre o qual SDN surgiu, podemos destacar a pesquisa em Redes Ativas (*Active Networks*). Desenvolvida em meados da década de 1990 e publicada por TENNENHOUSE et al. (1997), essa pesquisa trata de uma arquitetura de rede na qual os comutadores executam computação customizada nas mensagens que fluem através deles. Portanto, essa iniciativa propôs uma das primeiras ideias de uma infraestrutura de rede programável. A arquitetura é baseada em duas abordagens fundamentais: *switches* programáveis pelo usuário, com canais distintos de transferência de dados e gerenciamento, e que seriam capazes de processar rotinas customizadas; e cápsulas, que são fragmentos de programas que seriam carregados em mensagens de usuários e executados pelos dispositivos da rede. Apesar de promissora, essa proposta nunca foi adotada amplamente devido a questões práticas de segurança e desempenho (NUNES et al., 2014).

Outra iniciativa que merece destaque é a do grupo de trabalho *Open Signaling* (OPENSIG). Reunido a partir de 1995, esse grupo considerava que a separação entre o *hardware* de comunicação e o *software* de controle era necessária, embora fosse desafiadora devido a arquitetura dos *switches* e roteadores. A proposta do OPENSIG estava centrada em fornecer acesso ao *hardware* de rede através de interfaces abertas e programáveis. Movido por essas ideias, o IETF (*Internet Engineering Task Force*) formalizou um grupo de trabalho que levou à especificação do GSMP (*General Switch Management Protocol*), um protocolo de propósito geral para gerência de *switches*. O grupo de trabalho foi oficialmente concluído pelo IETF e a última proposta do protocolo (GSMPv3) (DORIA; SUNDELL, 2002) foi publicada em 2002 (NUNES et al., 2014).

Em uma abordagem posterior, o IETF propôs a arquitetura ForCES (*Forwarding and Control Element Separation*) (DORIA et al., 2010). Com sua padronização iniciada em 2003, ForCES separa os planos de dados e de controle através da definição de dois elementos funcionais: elemento de encaminhamento (*Forwarding Element* – FE) e elemento de controle (*Control Element* – CE) (MACEDO et al., 2015). Juntamente com outras arquiteturas desenvolvidas nos anos seguintes, ForCES é uma das principais iniciativas que seguiram na direção do desacoplamento dos mecanismos de controle e de encaminhamento de pacotes de dados, um dos pilares fundamentais do paradigma SDN.

O predecessor imediato de SDN foi o projeto Ethane (CASADO et al., 2007), o qual estabeleceu uma nova arquitetura para redes. Essa arquitetura foi projetada especificamente para abordar os problemas encarados pelas redes corporativas. O projeto fundamentava-se no uso de um elemento controlador externo logicamente centralizado para gerenciar políticas de segurança e de controle de acesso à rede baseado em identidade. Ethane empregava dois componentes: um controlador, para decidir se um pacote deveria ser encaminhado; e um *switch* Ethane, que mantinha uma tabela de fluxos e comunicava-se

com o controlador através de um canal de comunicação seguro. Como será discutido adiante, esse projeto elaborou várias das ideias e abordagens que viriam a ser adotadas posteriormente pela arquitetura SDN e pela tecnologia OpenFlow, como o conceito de fluxo e a transferência das decisões de encaminhamento para o controlador externo. De fato, no contexto do paradigma SDN, o controle de acesso baseado em identidade do Ethane estaria associado, nos dias de hoje, a uma aplicação de rede implementada em um controlador SDN atual (NUNES et al., 2014).

## 2.3 DEFINIÇÕES

Os projetos e tecnologias discutidos na Seção 2.2 buscaram tentar superar as restrições impostas pelas arquiteturas de redes tradicionais. Nesse contexto, os conceitos desenvolvidos a partir da combinação daquelas ideias e propostas sedimentaram a formação do novo paradigma SDN. O termo SDN surgiu primeiramente no âmbito da Universidade de Stanford (EUA). Entretanto, múltiplas definições para SDN vêm sendo elaboradas ao longo dos últimos anos.

Uma das descrições mais abrangentes é aquela apresentada pela *Open Networking Foundation* (ONF). ONF é um consórcio de empresas formado em 2011, sem fins lucrativos, cuja missão é promover a transformação das infraestruturas de redes e dos modelos de negócios nessa área (ONF, 2019). Acima de tudo, essa organização estimula a adoção do paradigma SDN e de tecnologias associadas a esse modelo. Segundo a definição da ONF, SDN é uma nova abordagem para redes de dados na qual o controle de rede é diretamente programável e dissociado das funções de encaminhamento de dados. Como resultado, cria-se uma arquitetura dinâmica, gerenciável, econômica e adaptável, que oferece aos administradores de redes enormes capacidades de programação, automação e controle (ONF, 2013).

Em uma definição mais técnica e concisa, SDN refere-se a uma arquitetura de rede fundamentada em quatro pilares: (i) desacoplamento dos planos de dados e de controle; (ii) transferência da lógica de controle para um elemento controlador externo logicamente centralizado; (iii) decisões de encaminhamento baseadas em fluxos; e (iv) programação da rede através de aplicações de *software* executando no topo do modelo (KREUTZ et al., 2015). Dentre esse conjunto de características fundamentais, algumas delas afetam diretamente a arquitetura dos dispositivos de redes tradicionais. Certamente, SDN é uma alternativa ao plano de controle proprietário e distribuído das abordagens de rede convencionais, permitindo aos administradores explorarem as verdadeiras capacidades das redes de comunicação.

## 2.4 ARQUITETURA SDN

A Figura 1 ilustra a arquitetura básica do paradigma SDN no contexto de um dispositivo de rede tradicional. A abordagem adotada por SDN separa o *software* de controle do dispositivo de rede subjacente. Nessa condição, um *software* SDN passa a ser executado em um controlador externo. A inclusão desse controlador na infraestrutura de comunicação estende a funcionalidade do plano de controle externamente ao equipamento de rede. Um novo módulo SDN é adicionado ao plano de controle do dispositivo, o que permite ao controlador externo configurar o plano de dados, atuando no processamento e no encaminhamento dos pacotes de dados. As decisões de encaminhamento são tomadas pelo controlador levando em consideração o conceito de fluxo. Um fluxo é um conjunto de pacotes de dados que possuem exatamente os mesmos campos de cabeçalho. O módulo SDN não fornece uma interface de gerenciamento convencional. Ao invés disso, o módulo simplesmente aceita comandos a partir do controlador, passando cada comando através da unidade de processamento do plano de dados. Nessa arquitetura, a interface original de controle e configuração torna-se secundária e muitas vezes desnecessária. Realmente, SDN aproveita o *hardware* de classificação e encaminhamento de dados, mas ignora o *software* de controle oferecido pelo fornecedor (COMER, 2015).

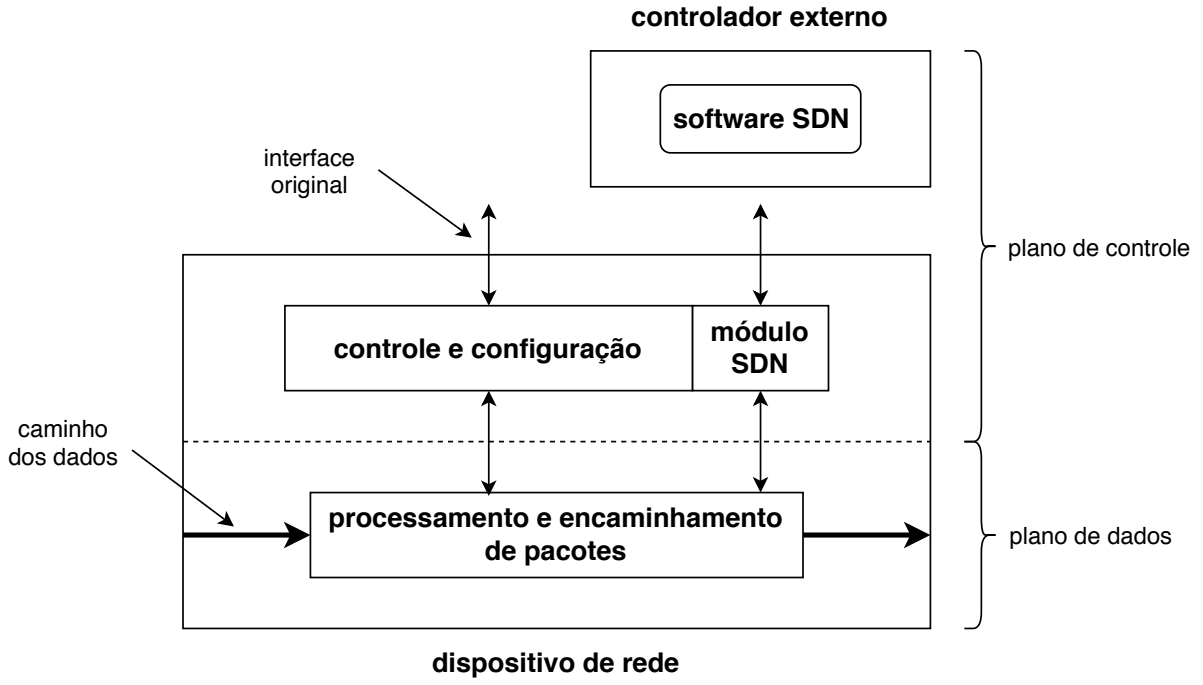



Figura 1 – Arquitetura básica SDN no contexto de um dispositivo de rede. Fonte: Adaptado de COMER (2015).

Para fornecer recursos significativos a uma rede, a tecnologia SDN deve ser expandida a vários dispositivos. Nesse sentido, a Figura 2 apresenta uma visão simplificada de uma arquitetura típica SDN em um contexto mais amplo, ou seja, considerando uma infraestrutura de rede completa. Nessa representação, os elementos de encaminhamento de

dados (comutadores e roteadores) que compõem a infraestrutura de rede têm seus planos de controle transferidos para o controlador externo, que está logicamente centralizado. Dessa forma, o controlador passa a ter uma visão unificada e global do estado da rede. Ao mover as funções do plano de controle para o controlador, o paradigma SDN fornece maior controle ao administrador da rede, simplificando a gerência do ambiente como um todo. Atualmente, existem várias implementações de controladores SDN. Alguns dos exemplos incluem o POX (POX, 2019), o Ryu (RYU, 2019), o Floodlight (FLOODLIGHT, 2019) e o OpenDaylight (OPENDAYLIGHT, 2019).

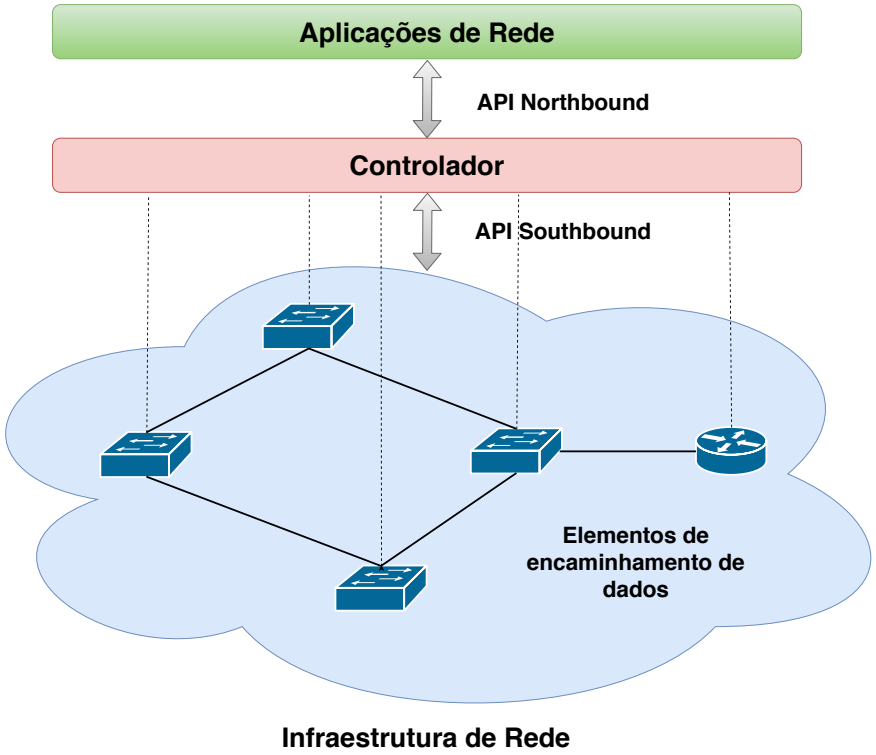



Figura 2 – Arquitetura típica SDN no contexto de uma infraestrutura de rede. Fonte: Adaptado de KREUTZ et al. (2015).

É importante destacar que o conceito de controlador de rede único só faz sentido sob o ponto de vista de uma representação lógica da arquitetura. O modelo SDN não exige, necessariamente, a figura de um só controlador como componente concentrador da rede, nem mesmo sugere sua localização física em um ponto específico do sistema. De fato, a presença de uma ou mais instâncias do controlador externo deve ser avaliada no ambiente de rede, principalmente em infraestruturas mais complexas e críticas, que não toleram pontos únicos de falha (PEIXOTO; VIEIRA, 2017; PEIXOTO et al., 2018). Questões relativas à localização, distribuição, interligação e balanceamento de carga de trabalho dos controladores dependem, dentre vários outros aspectos, da quantidade de componentes da rede, das aplicações SDN e dos requisitos de desempenho da comunicação de dados. Na realidade, muitas dessas questões ainda são discutidas.

A Figura 2 também mostra que a interação entre o controlador SDN e os dispositivos de redes se dá através de uma interface conhecida como API *Southbound*. A tecnologia OpenFlow (MCKEOWN et al., 2008) é um dos principais exemplos dessa API. Como será detalhado na próxima seção, OpenFlow oferece uma comunicação segura entre o controlador e os elementos de encaminhamento, sendo considerado atualmente um padrão *de facto* para a API *Southbound*. Por sua vez, a comunicação entre o controlador SDN e as aplicações de rede é realizada por intermédio da API *Northbound*. Essa API pode ser vista como uma interface de programação com um maior grau de abstração, o que permite a operadores e desenvolvedores criarem aplicações de rede em linguagens de alto nível, como Python, Java e C++. Ao contrário da API *Southbound*, a API *Northbound* ainda não é padronizada. Desse modo, a maioria dos controladores atuais oferece suas próprias APIs específicas.

Como retrata ainda a Figura 2, as aplicações de rede encontram-se no topo do modelo SDN. Por essa perspectiva, os controladores também são considerados como os sistemas operacionais das SDNs. As aplicações implementam o controle lógico que será traduzido em comandos. Esses comandos farão a inserção das regras no plano de dados, ditando o comportamento dos dispositivos (COSTA et al., 2016). Entre os muitos serviços e funcionalidades que podem ser programados na rede através dessas aplicações, podemos destacar os mecanismos de roteamento dinâmico, de engenharia de tráfego, de balanceamento de carga, de qualidade de serviço (QoS), de gerenciamento de energia e de virtualização, monitoramento e segurança de rede.

## 2.5 TECNOLOGIA OPENFLOW

Uma das principais premissas do paradigma SDN é a transferência da lógica de controle dos dispositivos de redes para o controlador externo. Nesse cenário, equipamentos como comutadores e roteadores devem oferecer recursos de configuração e controle ao controlador SDN. Da mesma forma, o controlador deve conhecer os respectivos atributos que o permitem fazer uso desses recursos. Assim, ao longo dos anos, foram propostas algumas tecnologias capazes de implementar meios de comunicação entre esses componentes. Contudo, uma dessas tecnologias está associada ao paradigma SDN desde a sua concepção, tornando possível o seu desenvolvimento e a sua implementação prática: o OpenFlow (COSTA et al., 2016).

Originalmente criado na Universidade de Stanford, o OpenFlow foi proposto por MCKEOWN et al. (2008) como uma maneira para pesquisadores executarem protocolos experimentais em redes tradicionais reais. Baseado na tecnologia Ethernet, sua utilização em um dispositivo de rede requer somente a instalação de um módulo OpenFlow. Na arquitetura SDN, o módulo OpenFlow corresponde ao módulo SDN mostrado na Figura 1, atuando apenas como um intermediário que traduz mensagens provenientes do controlador

externo em comandos que são transmitidos para o plano de dados do *hardware* do dispositivo de rede (COMER, 2015). Devido às suas características e funcionalidades, a tecnologia OpenFlow passou a ser utilizada pela maioria das plataformas SDN atuais. Hoje em dia, diversas empresas de tecnologia oferecem equipamentos de *hardware* habilitados com OpenFlow, entre elas grandes fornecedores de dispositivos de redes do mercado como Juniper, HP, Extreme, Huawey e Brocade (KREUTZ et al., 2015). Atualmente, também estão disponíveis comutadores em *software* com suporte a OpenFlow. Um dos principais exemplos é o Open vSwitch (OVS, 2019), um *software switch* bastante usado em implementações do paradigma, sendo inclusive utilizado nas avaliações do Capítulo 5.

Conforme mostra a Figura 3, um dispositivo OpenFlow (por exemplo, um *switch*) é composto de ao menos três partes: (i) uma Tabela de Fluxos, com uma ação associada a cada entrada na tabela, que orienta o dispositivo sobre como processar cada fluxo de pacotes de dados; (ii) um Canal Seguro, com possibilidade de uso do protocolo SSL, que conecta o dispositivo ao processo de controle remoto (controlador), permitindo que comandos e pacotes sejam enviados entre eles; e (iii) o Protocolo OpenFlow, que fornece uma interface aberta e padronizada para o controlador comunicar-se com o dispositivo, por meio de mensagens com formatos e representações específicas (MCKEOWN et al., 2008). Desde a sua criação, várias versões da tecnologia foram publicadas. De todo modo, independentemente da versão da especificação, o OpenFlow oferece uma solução acessível e eficiente para programar a tabela de fluxos em diferentes dispositivos de redes.

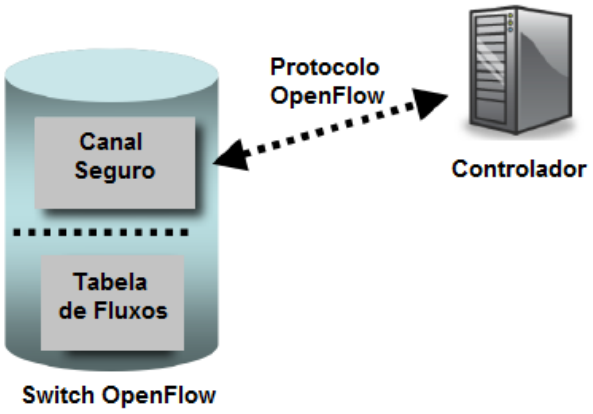



Figura 3 – Idealização de um *switch* OpenFlow. Fonte: COSTA et al. (2016).

A especificação básica do OpenFlow estabelece que a tabela de fluxos do dispositivo de rede implemente a classificação e o encaminhamento de pacotes de dados. Para tanto, a tabela é composta por um conjunto de entradas, onde cada entrada possui três partes: (i) um conjunto de padrões (Regras), que são confrontados com os pacotes e definem o fluxo associado a essa entrada; (ii) um campo de Ações, que determina como os pacotes do fluxo são tratados; e (iii) um conjunto de Estatísticas relacionadas à entrada, que inclui uma contagem do número de pacotes e *bytes*, além de uma marcação de tempo que indica a

última correspondência na entrada. A Figura 4 apresenta o formato geral de uma entrada na tabela de fluxos OpenFlow.

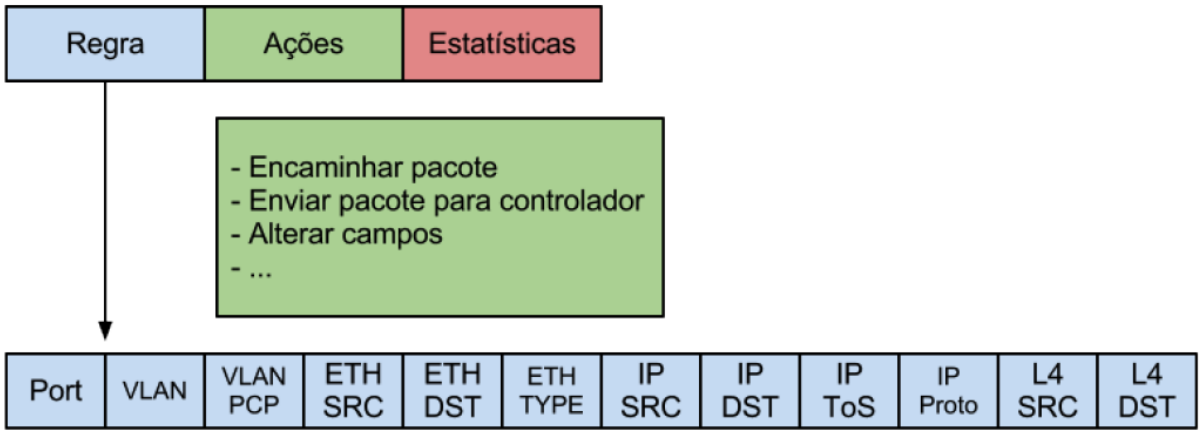



Figura 4 – Formato de uma entrada na tabela de fluxos OpenFlow. Fonte: GUEDES et al. (2012).

A Figura 4 destaca um conjunto básico de regras capaz de corresponder, entre outros, a portas e endereços de origem/destino de protocolos convencionais das camadas OSI de enlace (ETH), rede (IP) e transporte (L4). A capacidade de definição dos fluxos é limitada pelo conjunto de campos de uma implementação em particular, o qual estabelece a quantidade de arranjos possíveis. Como será discutido mais adiante nesta seção, esse conjunto varia de acordo com a versão do OpenFlow. Portanto, a classificação do pacote é determinada pela correspondência do padrão de cada entrada com os campos de cabeçalho do pacote. Caso haja uma equivalência em alguma entrada da tabela, o pacote é vinculado ao fluxo correspondente e processado conforme a ação indicada; caso nenhuma das entradas forneça uma correspondência, o pacote é encaminhado ao controlador para que o mesmo decida sobre a atitude a ser tomada em relação ao pacote e ao fluxo ao qual ele pertence. Nesse caso, após a decisão, o controlador instala uma nova entrada (regra) no dispositivo de rede, o qual passa a executá-la.

O encaminhamento efetivo dos pacotes de dados se dá a partir de uma ou mais ações determinadas pelas respectivas entradas da tabela de fluxos. A especificação básica do OpenFlow define três ações elementares que podem ser associadas com um padrão de classificação: (i) encaminhar o pacote para uma determinada porta ou conjunto de portas do dispositivo; (ii) encapsular o pacote e enviá-lo para o controlador para processamento, o que pode envolver também a instalação, pelo controlador, de uma nova regra na tabela para o tratamento dos próximos pacotes pertencentes ao mesmo fluxo; e (iii) descartar o pacote. Como mostra ainda a Figura 4, outras ações podem ser suportadas, como por exemplo a alteração de campos de cabeçalho, o enfileiramento de pacotes em filas de provisão de QoS e até mesmo o encaminhamento de pacotes através do *pipeline* de processamento padrão disponibilizado pelos fornecedores de dispositivos comerciais.

Conforme comentado anteriormente, existem diferentes versões do OpenFlow. A primeira versão estável (1.0) foi lançada em dezembro de 2009 e corresponde, basicamente, à especificação discutida nesta seção. No momento, a versão 1.0 é a mais amplamente utilizada e suportada, sendo também adotada nas avaliações descritas no Capítulo 5 desta dissertação. A versão 1.1, publicada em fevereiro de 2011, passou a suportar operações sobre o protocolo MPLS (*Multiprotocol Label Switching*) e em múltiplas tabelas de fluxos. Em dezembro de 2011 foi disponibilizada a versão 1.2, a qual acrescentou suporte aos protocolos IPv6 e ICMPv6 e a possibilidade de conectar um dispositivo de rede a múltiplos controladores simultaneamente. A versão 1.3 foi lançada em junho de 2012 e, entre outras melhorias, adicionou suporte à comunicação encriptada por TLS. A versão 1.3 também foi largamente aceita pela comunidade. Em outubro de 2013 foi especificada a versão 1.4, que trouxe diversas novidades como a possibilidade de execução de ações conjuntas e a exclusão de regras de menor importância em tabelas de fluxos totalmente ocupadas. Finalmente, em dezembro de 2014 foi lançada a especificação 1.5 do OpenFlow, sua atual versão. Entre outras funcionalidades, a versão 1.5 acrescentou suporte à correspondência por *flags* TCP como SYN, ACK e FIN (COSTA et al., 2016).

# 3 ECOSSISTEMAS DE COMPUTAÇÃO AVANÇADA

Este capítulo mostra os conceitos por trás dos ambientes de computação avançada. Devido a abrangência do tema, procuramos nos limitar aos aspectos que, de certa maneira, guardam alguma relação com o objetivo desta dissertação. Assim, a Seção 3.1 descreve as principais características das arquiteturas de computação paralela. Na Seção 3.2, são abordadas as tecnologias e arquiteturas de redes de comunicação usadas nos ambientes paralelos. Por fim, a Seção 3.3 traz algumas considerações sobre os modelos e *frameworks* de programação paralela.

## 3.1 ARQUITETURAS PARALELAS

Sistemas que demandam grande poder computacional, como aplicações científicas e muitas das atuais aplicações de ferramentas de negócios, utilizam arquiteturas de computação paralela para processarem seus dados no menor tempo possível. Segundo MATTSON et al. (2004), sistemas paralelos são criados combinando múltiplos elementos de processamento em um único sistema maior. Desde a metade dos anos 1990, tecnologias como *multithreading* e *multicore* tornaram esses sistemas amplamente disponíveis. O termo *multithreading* é usado para descrever a situação em que se permite a existência de múltiplos *threads* (fluxos) no mesmo processo, permitindo que múltiplas execuções ocorram no mesmo ambiente do processo com um grande grau de independência. O termo *multicore*, por sua vez, refere-se a um sistema que possui dois ou mais *cores* (núcleos) de processamento, possibilitando a execução de múltiplos processos em paralelo em um único computador (TANENBAUM, 2003).

A maneira mais comum para caracterizar as arquiteturas paralelas é a taxonomia de Flynn (FLYNN, 1972). Essa sistemática categoriza todos os computadores de acordo com o número (um ou múltiplos – *single* ou *multiple*) de fluxo de instruções (*instruction*) e de dados (*data*) que eles possuem, dividindo-os em quatro categorias: SISD, SIMD, MISD e MIMD. A maioria dos sistemas paralelos modernos encaixam-se na arquitetura MIMD (*Multiple Instruction, Multiple Data*), onde cada elemento de processamento tem seu próprio fluxo de instruções operando em seus próprios dados. A categoria MIMD é tipicamente classificada de acordo com a organização de memória: compartilhada e distribuída (MATTSON et al., 2004).

Em sistemas de memória compartilhada, todos os processos compartilham um único espaço de memória. Os processadores com múltiplos núcleos que equipam grande parte dos computadores atuais encaixam-se nessa categoria. Esses sistemas também são geralmente classificados em sistemas SMP (*Symmetric Multiprocessors*), onde todos os processadores acessam todas as posições de memória em velocidades iguais, e sistemas NUMA (*Non-Uniform Memory Access*), onde os tempos de acesso a determinadas posições

de memória podem ser significativamente diferentes, dependendo de quão próximo a posição da memória está do processador. Uma variação da classe NUMA é a ccNUMA (*cache-coherent* NUMA), na qual os efeitos do acesso não uniforme são minimizados com o uso de uma memória *cache* em cada processador (MATTSON et al., 2004).

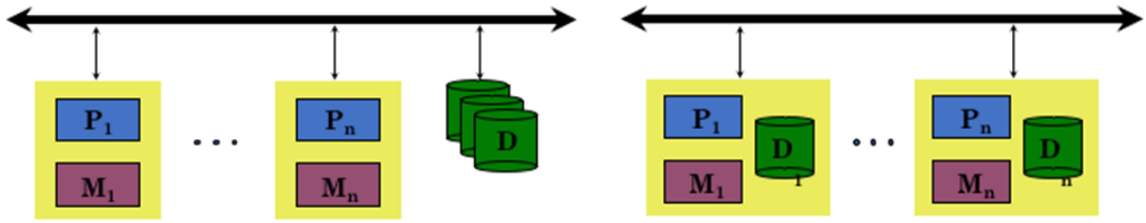
Sistemas de memória distribuída são aqueles onde cada processo tem seu próprio espaço de endereçamento e a comunicação com os demais processos ocorre por envio e recebimento de mensagens (*message passing*) via uma rede de comunicação. Esses sistemas são tradicionalmente divididos em duas categorias: MPP (*Massively Parallel Processors*), nos quais os processadores e a infraestrutura de rede são fortemente acoplados e especializados; e aglomerados (*clusters*) de computadores, que são sistemas compostos de nós de trabalho interconectados por uma tecnologia de rede, convencional ou específica (vide Seção 3.2), o que os tornam mais baratos e, portanto, cada vez mais comuns. Arquiteturas paralelas também podem apresentar classificações complementares: sistemas híbridos são *clusters* de nós onde cada nó contém vários processadores que compartilham memória; *grids* são sistemas que usam recursos heterogêneos e distribuídos, conectados por redes locais (LANs) e/ou de longa distância (WANs) (MATTSON et al., 2004).

As atuais arquiteturas de computação de alto desempenho (*High-Performance Computing* – HPC) possuem a tendência de migrar dos tradicionais sistemas SMP e MPP para sistemas de *cluster* de memória distribuída (DATE et al., 2016). Plataformas de supercomputação modernas geralmente extrapolam essa tendência, organizando-se em sistemas híbridos com uma configuração também híbrida de nós. Esse é o caso, por exemplo, do supercomputador brasileiro SDumont (LNCC, 2019). Nesse tipo de sistema, os nós de trabalho executam processos que usualmente compartilham um sistema de arquivos distribuídos como o Lustre (LUSTRE, 2019), um sistema de arquivos paralelos *open source* dotado de nós dedicados ao armazenamento e à gerência dos arquivos. Por sua vez, os recursos dos *clusters* HPC são compartilhados por ferramentas que garantem o controle de acesso a esses recursos de acordo com as políticas de administração do ambiente. Como exemplo, o Slurm (SLURM, 2019) é um sistema de agendamento de tarefas e gerenciamento de *cluster open source*, tolerante a falhas e altamente escalável, usado por muitas dessas plataformas, incluindo o próprio SDumont.

No ecossistema *Big Data*, os dados a serem processados, a princípio, não cabem na memória principal de um único computador. Assim, *clusters* também são vistos como a plataforma padrão para esses ambientes (PORTO, 2017). No entanto, suas aplicações típicas executam com base em um modelo no qual, diferentemente dos ambientes de HPC, os dados são distribuídos pelos nós de trabalho e os processos são alocados a esses nós, buscando assim minimizar a transferência de dados em disco. Para tanto, esses ambientes empregam frequentemente um sistema de arquivos distribuídos como o HDFS (*Hadoop Distributed File System*) (HDFS, 2019). O HDFS é facilmente portável e projetado para ser

executado em *hardwares* comuns e de baixo custo. Por isso, é altamente tolerante a falhas. HDFS fornece alta vazão no acesso a dados de aplicação e é adequado para aplicações que possuem grandes conjuntos de dados (tipicamente de *Gigabytes* a *Terabytes*). Ao contrário de sistemas como o Lustre, sua arquitetura é baseada na premissa de que mover a computação para perto dos dados é geralmente melhor do que movimentar os dados para onde a aplicação está sendo executada. Diante de tudo isso, considera-se que os arcabouços típicos de *Big Data* encaixam-se em uma arquitetura sem compartilhamento.

Considerando o que foi abordado nesta seção, o foco desta dissertação recai sobre as arquiteturas de memória distribuída e sem compartilhamento em plataformas de *clusters*. A Figura 5 ilustra essas arquiteturas de computação paralela no contexto desses ambientes, considerando os recursos de processador ($P$), memória ($M$) e disco ($D$) dos nós de trabalho. Os respectivos arranjos de $P$, $M$ e $D$ caracterizam cada modelo. Na arquitetura de memória distribuída (Figura 5a), cada nó dispõe de sua própria área de memória e processador, porém compartilha o acesso aos dados com os demais nós através do sistema de arquivos distribuídos (disco compartilhado). Por sua vez, na arquitetura sem compartilhamento (Figura 5b), cada nó mantém seus próprios recursos de processador, memória e disco. Nesse caso, o sistema de arquivos fornece uma visão integrada das partições de dados que são distribuídas pelos nós (PORTO, 2017). Na Seção 3.3, são detalhados os aspectos relacionados ao processamento paralelo nos *clusters*.




(a) Memória distribuída (b) Sem compartilhamento

Figura 5 – Arquiteturas de computação paralela. Fonte: Adaptado de PORTO (2017).

## 3.2 REDES DE COMUNICAÇÃO EM AMBIENTES PARALELOS

As redes de comunicação dos sistemas de *cluster* em ambientes de HPC conectam os nós de trabalho através de interconexões de alta vazão e baixa latência. Suas arquiteturas são caracterizadas pelo uso intensivo de numerosos dispositivos de comunicação, propiciando o aprimoramento dos fatores de escalabilidade, balanceamento de carga e tolerância a falhas. Essas redes são implantadas com diversas tecnologias de comutação de pacotes e sobre topologias físicas multicaminhos como *fat-tree* (em árvore) e Torus (em cubo), que oferecem ao mesmo tempo desempenho e redundância.

*Fat-tree* é uma das arquiteturas mais implantadas em redes de *data center* (ARAUJO et al., 2017). Redes *fat-tree* também são utilizadas em inúmeras plataformas de supercomputação, incluindo o Summit (ORNL, 2019), o supercomputador mais poderoso do mundo atualmente, e o SDumont. Proposta em 1985 por LEISERSON (1985), a arquitetura *fat-tree* possui a capacidade de fornecer múltiplas rotas entre os nós de trabalho a partir de uma abordagem hierárquica e em camadas, na qual *switches* são organizados geralmente em níveis como acesso (ou borda), distribuição (ou agregação) e núcleo (ou raiz). Uma *fat-tree* pode ser descrita através da notação *m-port, n-tree*, onde $m$ é o número de portas do *switch* e $n$ é a altura da árvore (quantidade de camadas). Nessa modelagem, o tamanho da rede é determinado em função de $m$ e $n$, o que acaba definindo o número de nós de trabalho ($m.(m/2)^{(n-1)}$) e o número de *switches* ($(2n-1).(m/2)^{(n-1)}$) (LIU et al., 2017). A distribuição desses elementos na topologia também estabelece a quantidade de enlaces na rede e, consequentemente, os múltiplos caminhos disponíveis. Um exemplo gráfico dessa arquitetura pode ser visto na Figura 6. A imagem exibe uma rede *fat-tree 6-port, 3-tree*. Nessa rede, os *switches* (representados por retângulos) dispõem de seis portas e a árvore possui três camadas, o que define uma topologia com 54 nós de trabalho (representados por círculos) e 45 *switches*, além, é claro, de diversos caminhos mínimos distintos entre os nós de trabalho.

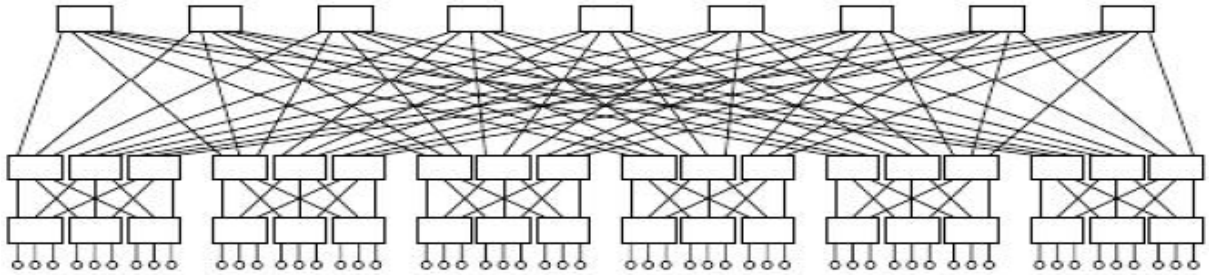


Figura 6 – Exemplo de rede *fat-tree 6-port, 3-tree*. Fonte: WEATHERSPOON (2014).

InfiniBand (INFINIBANDTA, 2019) é uma das principais tecnologias de comutação de pacotes utilizadas nos sistemas HPC. Com performance superior, ela provê a infraestrutura de comunicação de rede de mais de 25% das plataformas de supercomputação que compõem a lista de novembro de 2018 da TOP500[1], incluindo os dois primeiros colocados no *ranking*, os supercomputadores Summit e Sierra, respectivamente. Baseada em canal, InfiniBand é uma arquitetura de interconexão padronizada pela indústria desde 1999, capaz de prover conectividade de alto desempenho e baixa latência para servidores e *storages*. InfiniBand utiliza a tecnologia RDMA (*Remote Direct Memory Access*), que permite a movimentação de dados diretamente entre as memórias das aplicações, sem qualquer envolvimento de CPU ou sistema operacional. A especificação oferece múltiplos níveis de

[1] A lista TOP500 é um *ranking* periódico dos 500 supercomputadores mais poderosos do mundo. Sua edição mais recente está disponível em https://www.top500.org/lists/2018/11/ (Acesso em: Janeiro, 2019).

desempenho de *link*, com velocidades que atingem atualmente 100 Gb/s, suportando tanto cabeamento de cobre (até 30 m) quanto de fibra óptica (até 10 Km). InfiniBand também foi uma das primeiras especificações da indústria aptas a suportarem SDN.

Não menos importante, as redes Ethernet são amplamente utilizadas nos ecossistemas de computação avançada. Graças ao seu custo reduzido e à disponibilidade de produtos, fornecedores e especialistas, Ethernet é a tecnologia de comutação de pacotes predominante nos sistemas de *cluster* em ambientes *Big Data*, sobretudo em *clusters* de pequena escala. Mesmo com suas limitações de capacidade efetiva e retardo, essa tecnologia também é usada em ambientes de HPC, estando presente em praticamente 50% das plataformas listadas na última edição da TOP500. Nesse contexto, o paradigma SDN também passa a ser perfeitamente aplicável aos sistemas de computação paralela, principalmente através da tecnologia OpenFlow, a qual é intimamente ligada ao Ethernet. Na verdade, SDN complementa os modelos de rede tradicionalmente usados nos ambientes de computação paralela.

## 3.3 PROGRAMAÇÃO PARALELA

A classificação da arquitetura dos sistemas paralelos e as características das aplicações influenciam a escolha do modelo de programação usado para expressar a concorrência nas aplicações. Os modelos mais comuns são baseados nas arquiteturas de memória compartilhada, de memória distribuída com passagem de mensagem, ou em uma combinação híbrida das duas (MATTSON et al., 2004). O OpenMP (OPENMP, 2019) é um dos principais ambientes de programação para as arquiteturas de memória compartilhada. Formado por um conjunto de extensões de linguagem implementado como diretivas de compilador, o OpenMP é frequentemente usado para adicionar paralelismo ao código sequencial, particularmente em arquiteturas SMP.

O modelo de troca de mensagens entre processos é o modelo de programação paralela predominante para as aplicações que executam em arquiteturas de memória distribuída, especialmente nos ambientes de *clusters* HPC. O MPI (*Message Passing Interface*) (MPI, 2019) é o padrão de implementação mais usado nessas aplicações. Por definição, MPI é uma especificação de interface de bibliotecas de passagem de mensagem para desenvolvedores e usuários. Nesse modelo, um processo empacota informação em uma mensagem e a envia a um outro processo. Nas plataformas de *clusters*, os processos que executam em nós de trabalho distintos trocam mensagens através da rede de comunicação do *cluster*. MPI possui um conjunto de APIs de alto nível que fornece rotinas para gerenciamento de processos e operações de comunicação ponto-a-ponto e coletiva. Especificações de interface têm sido definidas para as linguagens C e Fortran. Criado nos anos 1990, sua especificação completa mais recente é a versão 3.1, embora já haja esforços para a padronização da versão 4.0. Atualmente, diversas bibliotecas implementam o padrão MPI, tanto abertas

como proprietárias. Open MPI (OPENMPI, 2019) e MPICH (MPICH, 2019) estão entre as implementações *open source* mais difundidas e utilizadas.

Nos ambientes *Big Data*, isto é, em arquiteturas sem compartilhamento, o modelo de programação funcional MapReduce (DEAN; GHEMAWAT, 2008) é o mais utilizado. Nele, usuários especificam a computação em termos de funções *map* e *reduce*. A função *map* processa cada item do conjunto de entrada (um par chave/valor), enquanto a função *reduce* processa um conjunto de elementos da entrada. A partir dessa simplificação, uma grande variedade de problemas pode ser resolvida fornecendo-se o comportamento para as respectivas funções. Com base nesse modelo, o sistema de tempo de execução subjacente automaticamente paraleliza a computação através do *cluster* de nós de trabalho, manipulando falhas e agendando a comunicação entre esses nós (PORTO, 2017). Hoje em dia, existem diversos arcabouços (*frameworks*) de computação intensiva de dados que implementam o modelo MapReduce, como por exemplo o Hadoop (HADOOP, 2019), o Spark (ZAHARIA et al., 2010; SPARK, 2019), e mais recentemente o Flink (FLINK, 2019), todos atualmente mantidos pela fundação Apache (APACHE, 2019).

O Hadoop é um *framework* que permite o processamento distribuído de grandes conjuntos de dados através de *clusters*, sendo projetado para escalar a até milhares de nós de trabalho. Lançado em 2008, sua última versão estável (3.2.0) foi publicada ainda no começo de 2019. Seus módulos combinam o modelo de programação simplificado MapReduce com o sistema de arquivos distribuídos HDFS. Hadoop também inclui um módulo de agendamento de tarefas e de gerenciamento de recursos do *cluster*. Esse módulo distribui as tarefas entre os nós de trabalho e as controla. Os nós que recebem as tarefas *map* são chamados de *mappers*. Por sua vez, os que recebem as tarefas *reduce* são chamados de *reducers*. As execuções das tarefas no *cluster* são independentes, porém um *reducer* precisa reunir as saídas geradas pelos *mappers*, em uma fase do processamento MapReduce que é conhecida como fase de embaralhamento (*shuffle*). No contexto da infraestrutura de comunicação, a fase *shuffle* é a que mais consome recursos da rede. Nas versões mais atuais do Hadoop, YARN (VAVILAPALLI et al., 2013) é o módulo padrão para desempenhar essas funções de gerência. Em uma instalação típica de Hadoop/YARN, um nó de trabalho é designado como *NameNode* (na nomenclatura HDFS) e outro é designado como *ResourceManager* (na nomenclatura YARN). Esses nós são conhecidos como mestres (*masters*). Os demais nós atuam como escravos (*slaves*), sendo designados ao mesmo tempo como *DataNode* e *NodeManager*, respectivamente nas nomenclaturas HDFS e YARN. Em *clusters* menores, a atribuição das funções dos *masters* a um único nó de trabalho também é viabilizada pelo *framework*.

# 4 PLATAFORMA DE COMUNICAÇÃO

Este capítulo explora a proposta de plataforma de comunicação baseada em SDN. Em primeiro lugar, a Seção 4.1 apresenta considerações iniciais importantes a respeito da motivação e dos problemas envolvidos no âmbito da confluência em computação paralela e distribuída. Em seguida, a Seção 4.2 aborda os aspectos funcionais da solução proposta. Em especial, essa seção fornece uma visão geral da plataforma SDN (Subseção 4.2.1), destacando as etapas e processos que implementam seus mecanismos essenciais, além de exibir uma representação de sua arquitetura. A Seção 4.2 ainda detalha os módulos da plataforma SDN (Subseção 4.2.2) e a API de programação fornecida (Subseção 4.2.3).

## 4.1 CONSIDERAÇÕES INICIAIS

*Big Data* vem se consolidando como um dos maiores fenômenos em termos de modelo de negócios e de ciência em quase todos os domínios (ECONOMIST, 2010). Empresas, órgãos públicos e instituições de pesquisa vêm se beneficiando amplamente das abordagens de *Big Data Analytics*, extraindo informações extremamente valiosas a partir dos grandes volumes de dados gerados no âmbito dessas organizações. Realmente, o uso de soluções de computação intensiva de dados tem se ampliado e crescido a cada dia, tornando-se indispensável em inúmeras áreas e instâncias, o que nos aproxima, efetivamente, da era da computação exaescala.

Manipular imensos volumes e diversidade de dados impõe enormes desafios. Conforme descrito no Capítulo 3, processar esses conjuntos de dados, estruturados ou não, por meio de aplicações típicas de *Big Data*, demanda a implantação de *clusters* formados por até milhares de nós de trabalho. Paralelamente, plataformas de HPC espalhadas ao redor do mundo dispõem de *clusters* com grande capacidade de processamento, sendo essencialmente empregadas na solução de aplicações científicas, em especial aquelas relacionadas à simulação numérica. Evidentemente, a implantação, operação e manutenção de ambientes distintos de computação avançada, para atender requisitos de aplicações também distintas, gera enormes custos e desperdiça recursos. De fato, implantações de plataformas de HPC consomem, sozinhas, investimentos da ordem de milhões de dólares. Da mesma forma, a grande escala das plataformas de processamento *Big Data*, mesmo composta por máquinas genéricas, eleva em muito o aporte técnico e financeiro direcionado a esses ambientes.

Nesse sentido, a confluência dos atuais ambientes de computação paralela e distribuída pode ser imaginada como um caminho óbvio na busca pela otimização de recursos e pela redução de custos, o que serve de motivação para a pesquisa de soluções eficientes nessa área. Sob o ponto de vista das plataformas de HPC, sua capacidade computacional pode ser aproveitada por uma vasta gama de aplicações, incluindo soluções para tratar grandes volumes de dados. Entretanto, apesar de a utilização de plataformas de HPC por

aplicações de *Big Data* pareça ser natural, o que se observa, de fato, é um descasamento entre essas arquiteturas. Assim, a integração entre esses mundos envolve o estudo e a adequação de vários elementos desses sistemas, desde a tecnologia de *hardware* até a programação em *software*, passando pela infraestrutura de comunicação de dados.

Como exemplo dessa problemática, a Figura 7 representa, em uma escala bastante reduzida, uma plataforma de comunicação que, no escopo de uma rede de dados tradicional, seria utilizada para implementar um ambiente de computação paralela e distribuída. Essa plataforma é tipicamente composta por um *cluster* de nós de trabalho conectados a dispositivos de comutação (*switches*) que, por sua vez, são interligados através de uma topologia física multicaminhos qualquer. Nesses ambientes, as aplicações *Big Data* e HPC são executadas em paralelo e de forma distribuída pelos nós da arquitetura, os quais trocam dados e/ou mensagens através da rede. Os fluxos de pacotes são encaminhados de acordo com políticas de roteamento configuradas de forma estática pelo administrador do ambiente. Além disso, a despeito do eventual emprego de mecanismos de segmentação lógica, o tráfego proveniente de cada aplicação é tratado de forma idêntica pelos *switches*. A princípio, esse cenário de rede compromete o desempenho geral das aplicações.

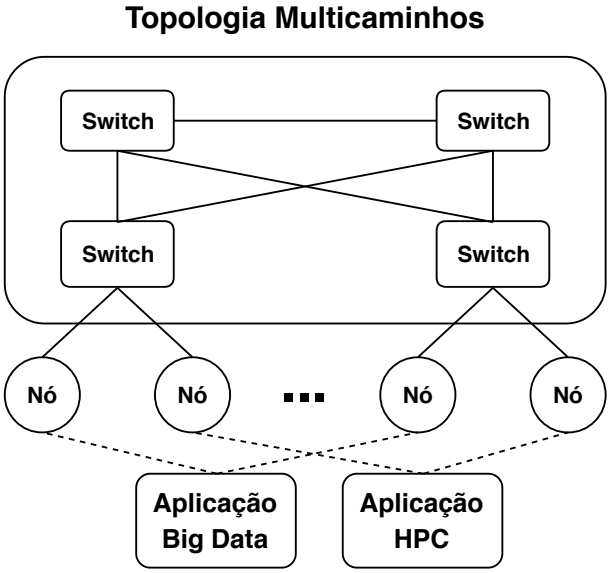



Figura 7 – Plataforma de rede tradicional para ambientes de computação paralela e distribuída.

Por outro lado, no âmbito de uma plataforma verdadeiramente integrada de computação paralela e distribuída, a infraestrutura de comunicação de dados precisa ser eficiente e flexível para se adequar aos diferentes perfis de tráfego gerados pelas aplicações que operam sobre a rede. Isso pressupõe o atendimento dos requisitos de rede de aplicações MPI, típicas de plataformas de HPC, e de aplicações MapReduce, típicas de ambientes de *Big Data*. Infelizmente, como visto no Capítulo 2, as redes de dados tradicionais possuem controles complexos, o que as tornam "ossificadas", ou seja, inflexíveis. A arquitetura dos dispositivos de redes convencionais exige configurações manuais e individualizadas

desses equipamentos. Logo, o ajuste das redes de comunicação aos requisitos de cada uma dessas aplicações demanda um alto custo administrativo, o que inviabiliza, na prática, o uso dessas infraestruturas de forma integrada e eficiente. Nesse contexto, a flexibilidade e o maior nível de programabilidade do paradigma SDN possibilitam o ajuste dinâmico das redes, o que pode favorecer, consequentemente, a efetiva integração dos ambientes de *Big Data* e de HPC.

## 4.2 SOLUÇÃO PROPOSTA

Diante dessas considerações, nossa proposta de plataforma de comunicação é idealizada sobre uma arquitetura fundamentada em SDN. A solução concebida é capaz de suprir, de forma flexível e dinâmica, os requisitos de desempenho das aplicações típicas de *Big Data* e de HPC. A plataforma proposta atende esses requisitos mediante a otimização da comunicação sob um modelo de rede ciente da aplicação. Nesse processo, a vazão da rede é maximizada e o tempo de execução das aplicações tende a diminuir.

Considerando como arquitetura de referência a mesma plataforma exibida na Figura 7, isto é, uma arquitetura de *cluster* com uma topologia física multicaminhos qualquer, o paradigma SDN é usado no aprimoramento da comunicação dos dados aproveitando os múltiplos caminhos disponíveis na infraestrutura de rede. A abordagem proposta apoia-se, essencialmente, na configuração dinâmica de políticas de roteamento mais adequadas aos perfis de tráfego de cada aplicação. Para tanto, o controlador SDN identifica o tráfego das aplicações e seleciona, de forma automatizada e eficiente, as melhores rotas para os respectivos fluxos de pacotes de dados, levando em conta as políticas de roteamento definidas. Nessa solução, o comportamento da rede é ditado com base no reconhecimento do tráfego, o que denota um modelo onde a rede torna-se ciente da aplicação. Como será abordado no Capítulo 6, essa abordagem constitui uma alternativa simples e eficiente àqueles mecanismos de otimização de rede que empregam conceitos de provisão de QoS. Embora também sejam eficientes, esses mecanismos compreendem funções específicas como controle de admissão, negociação de recursos e alocação de filas de QoS, que adicionam complexidade ao sistema e comprometem o atendimento dos requisitos de alto desempenho das plataformas de computação paralela e distribuída.

### 4.2.1 **Visão Geral da Plataforma SDN**

Em uma visão macro, a solução proposta pode ser compreendida a partir do encadeamento das etapas e processos que implementam a plataforma SDN. Desse modo, a Figura 8 mostra um diagrama básico que contém o fluxo de processamento do tráfego de rede e o relacionamento dessas etapas e processos no âmbito do ambiente integrado. No diagrama, os retângulos representam os processos, os cilindros "recortados" simbolizam os dados armazenados pela plataforma e o losango expressa a seleção da aplicação. As setas

finas retratam a sequência do processamento, enquanto que as setas grossas caracterizam a produção dos dados. Por sua vez, as linhas tracejadas refletem o consumo dos dados pelos processos.

Como condição inicial, a plataforma identifica o tráfego mediante a análise dos dados referentes às aplicações, dados estes fornecidos por elas próprias ou pelo administrador da rede. A princípio, no contexto da infraestrutura integrada, o tráfego pode conter pacotes de aplicações *Big Data* e HPC de forma concorrente. A partir dessa identificação, a plataforma seleciona uma rota dentre os múltiplos caminhos mínimos disponíveis na topologia. Essa seleção está associada ao tipo de aplicação que gerou o tráfego. Assim, o critério de escolha da melhor rota leva em conta as estratégias de roteamento definidas para o tráfego daquela aplicação, cujos dados são fornecidos pelo administrador do ambiente. Por fim, a plataforma providencia o encaminhamento do tráfego através da rota selecionada. O diagrama da Figura 8 também aponta que a descoberta dos caminhos na topologia e o processamento dos dados fornecidos são executados por processos autônomos e independentes do fluxo de tratamento dos pacotes.

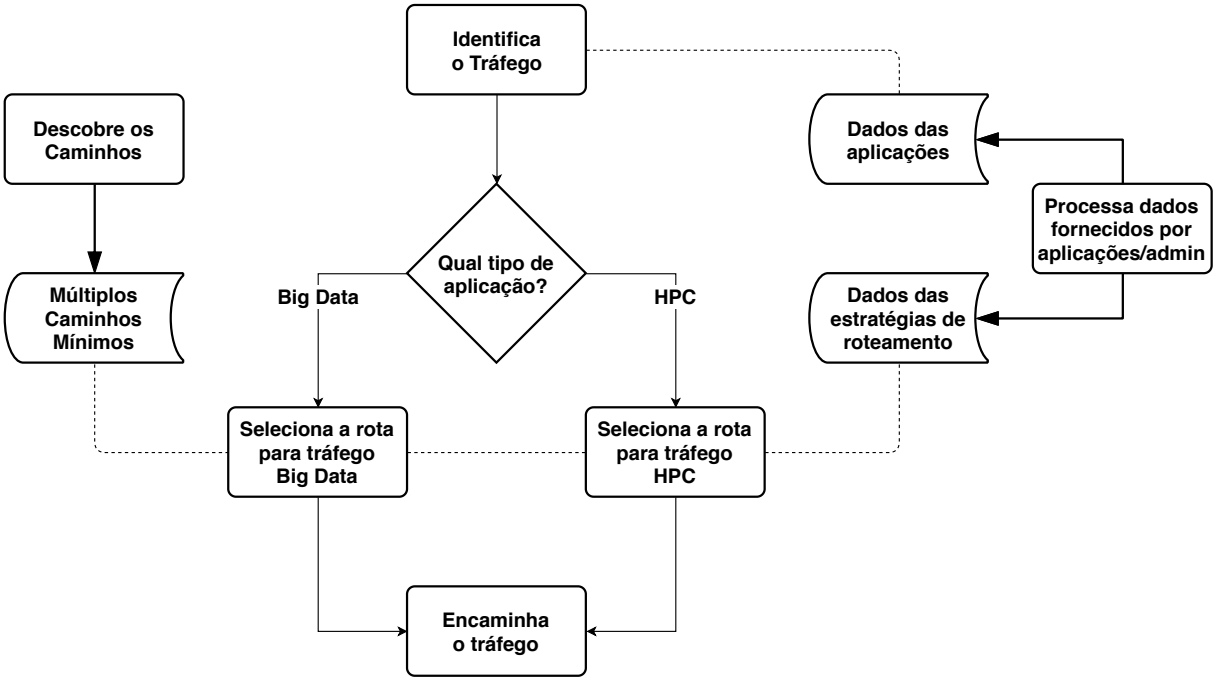



Figura 8 – Diagrama de etapas e processos que compõem a solução proposta para a plataforma SDN.

No contexto do paradigma SDN, as funcionalidades correspondentes às etapas e processos descritos logo acima são implementadas por intermédio de componentes dispostos em uma arquitetura típica do paradigma. Nesse sentido, a Figura 9 exibe uma representação, em alto nível, da arquitetura da plataforma SDN para ambientes de computação paralela e distribuída. Essa ilustração reflete, acima de tudo, a evolução da plataforma de rede tradicional da Figura 7, que passa agora a ser composta por *switches*

habilitados por SDN e gerenciada por um controlador compatível. Sua implantação é baseada em módulos de *software* de rede que executam no âmbito do controlador SDN e que implementam os mecanismos essenciais da plataforma.

Seguindo a abordagem estabelecida pela solução proposta, os principais componentes da arquitetura apresentada na Figura 9 são aqueles relacionados aos módulos SDN de descoberta de caminhos, de identificação do tráfego e de seleção de rotas. Os dados das aplicações e das estratégias de roteamento são fornecidos com a ajuda de uma API de programação disponibilizada pela plataforma, os quais são processados por um módulo próprio. As aplicações podem invocar as funções da API diretamente para informarem ao controlador SDN as características gerais dos seus perfis de tráfego. De modo similar, o administrador do ambiente faz uso direto da mesma API através de aplicações administrativas para fornecer os dados de configuração necessários. Módulos de encaminhamento de pacotes e de coleta de estatísticas de rede implementam as demais funcionalidades da plataforma de comunicação. Não obstante, outros módulos podem ser acrescentados ao controlador para adicionar eventuais novas funcionalidades. Na Figura 9 ainda é possível observar o *cluster* de nós de trabalho onde são executadas as aplicações *Big Data* e HPC sobre a topologia multicaminhos. A subseção a seguir explora esses módulos com maior profundidade.

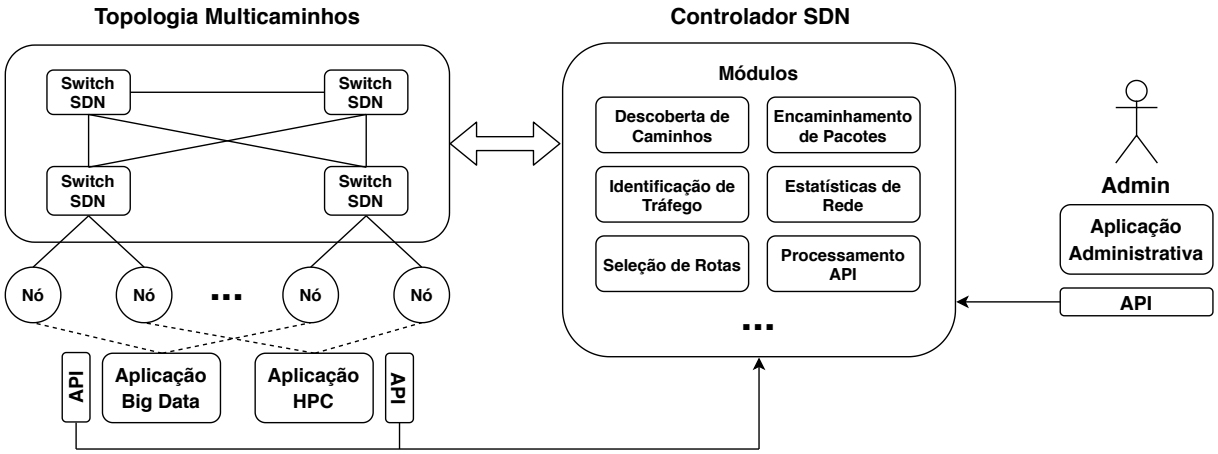



Figura 9 – Arquitetura da plataforma SDN para ambientes de computação paralela e distribuída.

### 4.2.2 **Módulos da Plataforma SDN**

O módulo de descoberta de caminhos deve fornecer uma perspectiva, em tempo real, de todos os enlaces de rede disponíveis na infraestrutura. Na topologia multicaminhos da plataforma, os nós de trabalho contam, quase sempre, com múltiplas rotas entre eles. No contexto da solução proposta, o interesse restringe-se ao descobrimento dos múltiplos caminhos mínimos entre esses nós. Para isso, podem ser usados algoritmos de roteamento de estado de enlace comuns. A visão global oferecida pelo paradigma SDN simplifica a

implementação desses algoritmos, tornando-os bastante eficientes. Dessa forma, permite-se criar uma visualização lógica da topologia de rede, a qual é usada pelo módulo de seleção de rotas.

O módulo de seleção de rotas implementa o principal fundamento da plataforma SDN proposta. Através dele, a plataforma aplica, de forma automatizada e eficiente, políticas de roteamento mais adequadas aos diferentes fluxos de pacotes de dados que trafegam sobre a rede. Dessa maneira, tráfegos de aplicações distintas podem ser tratados de modo distinto e mais apropriado. Os algoritmos de roteamento podem implementar diversas técnicas de engenharia de tráfego, encaminhando os fluxos por caminhos menos congestionados, por caminhos isolados ou simplesmente balanceando o tráfego na rede. Assim, a topologia lógica pode ser alterada em cada caso e para cada fluxo.

Diferentes aplicações MapReduce e MPI ajustam-se melhor a diferentes estratégias de roteamento em diferentes cenários. Aplicações que utilizam o modelo MapReduce possuem um padrão de comunicação bem definido, com destaque para a movimentação intensiva de dados na fase de embaralhamento (fase *shuffle*). No modelo MPI, o padrão depende da aplicação, com operações de comunicação coletiva geralmente prevalecendo sobre operações ponto-a-ponto, e problemas de granularidade fina demandando mais recursos de rede do que problemas de granularidade grossa[1]. Portanto, a definição e o emprego dos métodos de seleção de rotas mais adequados a cada padrão de comunicação influenciam diretamente no desempenho da plataforma. Em vista disso, na nossa abordagem, é essencial que o administrador da rede estabeleça as melhores alternativas para cada caso, levando em conta o ambiente como um todo. Avaliações dessas aplicações em função de diferentes estratégias de roteamento justificam as potenciais escolhas por parte do administrador.

A identificação do tráfego pode seguir uma abordagem tanto reativa quanto proativa. No modo reativo, as aplicações informam suas características gerais ao controlador SDN em tempo de execução, por intermédio da API de programação disponibilizada pela plataforma. Embora exija modificações nas aplicações, essa abordagem oferece grande flexibilidade aos usuários do ambiente. No modo proativo, o controlador SDN reconhece o tráfego de forma transparente, através de padrões comuns dessas aplicações, fornecidos previamente pelo administrador da rede mediante a mesma API e configurados pelo módulo próprio. Ambos os modos implementam um modelo onde a rede se torna ciente da aplicação, por meio de mecanismos de classificação de tráfego minimamente custosos computacionalmente.

O módulo de encaminhamento de pacotes implementa a função básica SDN de instalação das regras de encaminhamento nas tabelas de fluxos de cada dispositivo de rede. Essas regras executam, de forma concreta, a lógica da plataforma proposta, direcionando o

[1] Em computação paralela, granularidade é a medida qualitativa da razão entre a computação e a comunicação.

tráfego de rede por um caminho mínimo selecionado segundo uma política de roteamento previamente definida. O módulo de estatísticas de rede, como o próprio nome diz, realiza a coleta frequente de informações estatísticas de *bytes* e/ou pacotes de dados que trafegam pelas portas dos dispositivos de encaminhamento. Entre outras serventias, esse módulo é útil na identificação dos enlaces menos congestionados, que servem à implementação de estratégias de roteamento que consideram tal fator. Por fim, o módulo de processamento da API é responsável por tratar os dados provenientes das aplicações paralelas e administrativas, além de gerenciar a comunicação entre essas aplicações e o controlador SDN.

### 4.2.3 API de Programação

Seguindo os princípios e abordagens da solução proposta, idealizamos uma API básica a ser fornecida pela plataforma e que atende as funcionalidades previstas pela mesma. Para a finalidade de identificação de tráfego, a API é fundamentada na simples associação entre o tipo da aplicação (i.e., Hadoop, MPI, entre outras) e os números de portas dos protocolos da camada de transporte usados por cada uma delas. A API também permite que o administrador do ambiente defina o mapeamento entre o tipo da aplicação e o algoritmo de roteamento correspondente, provendo as informações necessárias à seleção de rotas. Para a associação entre os tipos de aplicações e suas portas, a API concebida propõe ao menos três funções. Essas funções podem ser invocadas diretamente pelas aplicações (modo reativo) ou pelo administrador do ambiente (modo proativo). São elas:

- **addAppData(app_type, port_ini, port_fin)**: Atribui um tipo de aplicação a um intervalo de números de portas de protocolos da camada de transporte usado para a comunicação dos dados. *app_type* é uma referência a uma aplicação *Big Data* ou HPC tal como '*hadoop*', '*mpi*' ou qualquer outra que seja reconhecida pela plataforma. *port_ini* é a primeira porta do intervalo. *port_fin* é a última porta do intervalo.

- **removeAppData(port_ini, port_fin)**: Remove a associação de um tipo de aplicação a seu intervalo de números de portas de protocolos da camada de transporte. *port_ini* é a primeira porta do intervalo. *port_fin* é a última porta do intervalo.

- **clearAppData()**: Exclui todas as associações entre os tipos de aplicações e seus intervalos de números de portas de protocolos da camada de transporte.

Para o mapeamento entre os tipos de aplicações e seus algoritmos de roteamento, a API idealizada propõe ao menos duas funções. Essas funções são usadas pelo administrador do ambiente. São elas:

- **setAlgAppMapping(app_type, alg)**: Mapeia um tipo de aplicação à sua estratégia de roteamento mais adequada no ambiente de execução. *app_type* é uma referência a uma aplicação *Big Data* ou HPC tal como '*hadoop*', '*mpi*' ou qualquer outra que seja reconhecida pela plataforma. *alg* é uma referência a um algoritmo de roteamento tal como '*stp*', '*ecmp*' ou qualquer outro que seja implementado na plataforma.

- **setAlgDefault(alg)**: Configura o algoritmo de roteamento padrão a ser aplicado aos fluxos de pacotes de dados para os quais não existe mapeamento previamente definido. *alg* é uma referência a um algoritmo de roteamento tal como '*stp*', '*ecmp*' ou qualquer outro que seja implementado na plataforma.

# 5 AVALIAÇÕES

Este capítulo apresenta a descrição completa das três avaliações executadas sobre a plataforma proposta. A Seção 5.1 aborda o cenário de uso e os algoritmos de roteamento considerados nos testes. Na Seção 5.2, destacamos a metodologia geral de análise. Nas seções subsequentes, são detalhadas as metodologias de avaliação específicas e os resultados obtidos em cada experimentação. Dessa forma, a Seção 5.3 explora a avaliação de rede com tráfego HPC. Na Seção 5.4, mostramos a avaliação de rede com tráfego *Big Data*. Por último, na Seção 5.5, discutimos a avaliação de desempenho de aplicações reais.

## 5.1 CENÁRIO CONSIDERADO

A Figura 10 apresenta o cenário de uso considerado em todas as avaliações desta dissertação. Ele representa uma plataforma de comunicação baseada em SDN, formada por um *cluster* de pequena escala com uma topologia física multicaminhos *fat-tree 4-port, 3-tree*, isto é, com *switches* de quatro portas e uma árvore de três níveis (de baixo para cima: acesso, distribuição e núcleo). A rede comutada possui 20 *switches* ($s1..s20$) gerenciados por um controlador SDN (não exibido na Figura 10). O cenário ainda é composto por 16 nós de trabalho (*hosts* $h1..h16$) e um servidor ($srv1$), o qual exerce o papel de servidor de arquivos distribuídos que pode ser compartilhado pelos nós. Nessa ilustração também é possível observar a representação dos quatro PODs (*Point of Delivery*).

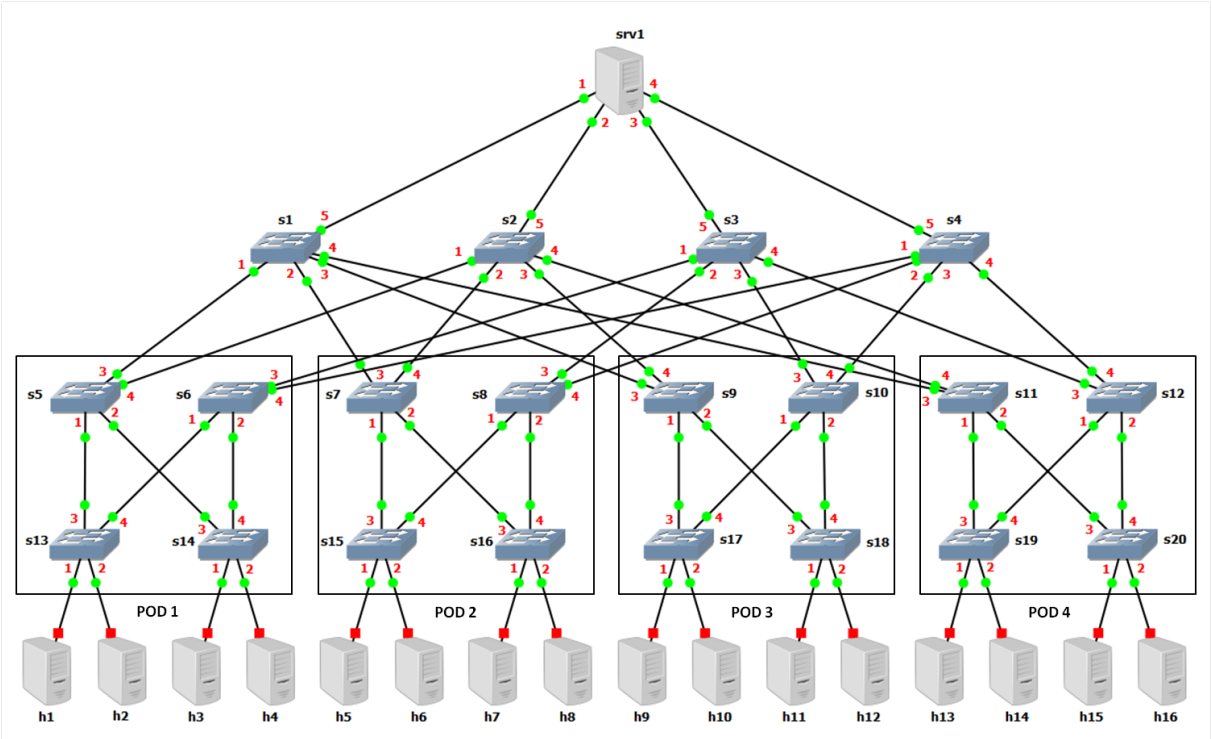



Figura 10 – Cenário de avaliação considerado.

Esse cenário retrata um ambiente reduzido que, de acordo com análises prévias, é capaz de capturar todos os aspectos relevantes às nossas avaliações. De fato, topologias *fat-tree* similares em tamanho são utilizadas com êxito em trabalhos relacionados como os de TAKAHASHI et al. (2018) e BHATIA et al. (2017). Sua arquitetura de camada 2 atende os requisitos de um ambiente típico HPC, tendo em vista que os *hosts* formam um sistema de memória distribuída, compartilhando os dados através de $srv1$. Ao mesmo tempo, o cenário encaixa-se em uma arquitetura típica de ambientes *Big Data*, ou seja, sem compartilhamento, levando em conta que os *hosts* também possuem capacidade própria de armazenamento. A comunicação de dados dispõe, via de regra, de múltiplos caminhos entre os pares de nós de trabalho. O número de caminhos mínimos entre esses pares varia em relação às suas posições na rede: nós conectados ao mesmo *switch* de acesso não possuem caminhos mínimos redundantes; nós conectados a *switches* de acesso diferentes dentro do mesmo POD dispõem de dois caminhos mínimos entre eles; nós conectados a PODs distintos possuem quatro caminhos mínimos diferentes.

Para estabelecer as melhores configurações de rede para as aplicações *Big Data* e HPC dentro desse cenário considerado, avaliamos os respectivos tráfegos a partir de quatro métodos simples de seleção de rotas. Eles são idealizados com a finalidade de maximizar a comunicação dos dados, enquanto tentam minimizar o custo computacional. Em termos gerais, as políticas de roteamento são baseadas em algoritmos comumente utilizados em redes de comutação de pacotes com múltiplos caminhos, e em trabalhos publicados no domínio das redes de *data center*, como por exemplo o artigo de WEBB et al. (2011), que será descrito no Capítulo 6. Assim, no cenário de avaliação, consideramos os seguintes algoritmos de roteamento:

• **"stp"**: O Algoritmo 1, aqui referenciado como "stp", seleciona sempre o mesmo caminho (por exemplo, o primeiro) dentre o conjunto de caminhos mínimos disponíveis entre uma origem e um destino. Ele é inspirado no protocolo *spanning tree* no sentido de que ignora as demais rotas, considerando-as como redundantes. Seu custo computacional é baixo, porém desperdiça os múltiplos caminhos da topologia. Um pseudocódigo é exibido adiante:

**Algoritmo 1:** "stp"

**Entrada:** Conjunto de caminhos mínimos
**Saída:** Primeiro caminho mínimo
1 **início**
2 | caminho mínimo ← primeiro caminho
3 **fim**
4 **retorna** *caminho mínimo*

• **"traffic"**: O Algoritmo 2, aqui referenciado como "traffic", seleciona o caminho mínimo menos congestionado entre uma origem e um destino. Para tal, dentre o conjunto de caminhos mínimos, é calculada a menor taxa de tráfego total instantânea dos seus enlaces.

Nesse cálculo, é considerado, por exemplo, o número de *bytes* enviados e recebidos na porta de saída de cada comutador pertencente ao caminho mínimo. Seu custo computacional é alto, entretanto usufrui das rotas com maior largura de banda disponível. A seguir, é mostrado um pseudocódigo para esse algoritmo:

**Algoritmo 2:** "traffic"

```
   Entrada: Conjunto de caminhos mínimos
   Saída: Caminho mínimo menos congestionado
 1 início
 2 |   caminho mínimo ← primeiro caminho
 3 |   taxa total menor ← ∞
 4 |   para cada caminho ∈ caminhos mínimos faça
 5 |   |   taxa do caminho ← 0
 6 |   |   para cada comutador ∈ ao caminho faça
 7 |   |   |   taxa do comutador ← bytes enviados/recebidos na porta de saída
 8 |   |   |   taxa do caminho ← taxa do caminho + taxa do comutador
 9 |   |   fim
10 |   |   se taxa do caminho ≤ taxa total menor então
11 |   |   |   taxa total menor ← taxa do caminho
12 |   |   |   caminho mínimo ← caminho
13 |   |   fim
14 |   fim
15 fim
16 retorna caminho mínimo
```

• **"ecmp"**: O Algoritmo 3, aqui referenciado como "ecmp", seleciona o caminho mínimo através de uma função de *hash*. A chave da função pode ser formada por uma n-tupla composta por campos de cabeçalho do fluxo de pacotes —por exemplo, (IP de origem, IP de destino, ID do protocolo)— ou pelo próprio valor individual do campo identificador (ID) do fluxo. Tais informações são obtidas pela API OpenFlow. Para mapear a chave a um índice associado ao caminho, pode ser utilizada uma função simples como a de módulo. Seu custo computacional é baixo e os multicaminhos são bem aproveitados, porém o uso de funções de *hash* não perfeitas pode gerar colisões e, consequentemente, caminhos ociosos. Esse algoritmo é inspirado no algoritmo ECMP (*Equal-Cost MultiPath*) (THALER; HOPPS, 2000). Um exemplo de pseudocódigo para o algoritmo "ecmp" é apresentado a seguir:

**Algoritmo 3:** "ecmp"

```
   Entrada: Conjunto de caminhos mínimos
   Saída: Caminho mínimo por hash
1 início
2 |   índice do caminho ← ID do fluxo % nº de caminhos mínimos
3 |   caminho mínimo ← caminhos mínimos [índice do caminho]
4 fim
5 retorna caminho mínimo
```

• **“isolated”**: O Algoritmo 4, aqui referenciado como “isolated”, é uma variação do Algoritmo 3 (“ecmp”). Ele seleciona um caminho mínimo disponível por meio de uma função de *hash* e o “isola” logica e temporariamente, inserindo-o, por exemplo, em um conjunto de caminhos isolados. Dessa forma, a rota permanece exclusiva para o fluxo, ficando indisponível para seleção pelos demais fluxos. Seu custo computacional também é baixo. Um pseudocódigo para esse algoritmo é descrito a seguir:

**Algoritmo 4:** “isolated”

```
   Entrada: Conjunto de caminhos mínimos
   Saída: Caminho mínimo isolado
 1 início
 2     enquanto ∃ caminhos mínimos faça
 3         índice do caminho ← ID do fluxo % nº de caminhos mínimos
 4         caminho mínimo ← caminhos mínimos [índice do caminho]
 5         se caminho mínimo ∉ caminhos isolados então
 6             caminho mínimo ∪ caminhos isolados
 7             vai para 14
 8         fim
 9         senão
10             caminhos mínimos ← caminhos mínimos - caminho mínimo
11         fim
12     fim
13 fim
14 retorna caminho mínimo
```

## 5.2 METODOLOGIA GERAL

Para avaliar a viabilidade e a aplicabilidade da plataforma SDN como um todo, necessitamos determinar a melhor configuração de rede para cada tipo de aplicação nesse ambiente integrado. Assim, de forma isolada, avaliamos inicialmente o desempenho de rede da plataforma proposta com um tráfego HPC. Em seguida, sob as mesmas condições, analisamos o desempenho de rede com um tráfego *Big Data*. O objetivo dessas avaliações foi identificar os métodos de roteamento mais adequados a cada tipo de tráfego. A partir dos resultados dessas análises no ambiente de testes, aplicamos esses métodos de roteamento em uma última avaliação, dessa vez conjunta, medindo o desempenho de aplicações reais. Diante desse planejamento, estabelecemos os critérios gerais sobre os quais todas as avaliações foram executadas.

Todas as avaliações foram realizadas por meio de simulações de um ambiente SDN Ethernet. Implementamos o cenário de rede descrito na Seção 5.1 no emulador Mininet versão 2.3.0d1 (MININET, 2019). O Mininet é um emulador de rede virtual *open source* proposto em 2010 por LANTZ et al. (2010) e nativamente preparado para reproduzir arquiteturas SDN e OpenFlow. Ele permite a prototipagem rápida de grandes redes em sistemas computacionais restritos, até mesmo em um único *laptop*. Mininet usa recursos

de virtualização a nível de sistema operacional, incluindo processos e *namespaces* de rede, permitindo-o escalar a até centenas de nós (*switches*, *hosts* e controladores). Esses recursos fornecem processos individuais com interfaces de rede, tabelas de roteamento e tabelas ARP separadas. Na SDN emulada, os comutadores executaram o Open vSwitch 2.0.2 e os enlaces foram limitados a 1 Gbps. Como controlador SDN, empregamos uma única instância do POX *carp* utilizando o OpenFlow 1.0 e operando remotamente em relação ao Mininet. A execução do POX deu-se dentro do mesmo servidor.

Para a execução dos testes no ambiente emulado, desenvolvemos os protótipos dos módulos de *software* da plataforma abordados no Capítulo 4. Entre as principais características desses protótipos, algumas merecem destaque. Implementamos o módulo de descoberta de caminhos mais curtos a partir do algoritmo de estado de enlace de Dijkstra (DIJKSTRA, 1959). Por sua vez, o desenvolvimento dos módulos de identificação de tráfego e de seleção de rotas seguiu os princípios definidos pela API descrita no Capítulo 4 e pelos algoritmos de roteamento citados na Seção 5.1. A rotina de processamento da API utiliza como base os módulos nativos do POX *messenger* e *messenger.tcp_transport*. Da mesma forma, os módulos POX *openflow.discovery* e *host_tracker* também fornecem suporte a alguns dos protótipos desenvolvidos, entre eles o módulo de encaminhamento de pacotes. O módulo de estatísticas de tráfego de rede realiza a coleta das informações com um intervalo padrão de 1 segundo. Além dos módulos do controlador SDN, elaboramos ainda um *script* de geração de topologias *fat-tree* genéricas para o Mininet. Toda a programação foi feita usando a linguagem Python 2.7 (PYTHON, 2019).

## 5.3 AVALIAÇÃO DE REDE COM TRÁFEGO HPC

Considerando os critérios gerais descritos na Seção 5.2, procuramos determinar a melhor configuração de rede para as aplicações típicas de HPC. Para isso, analisamos um tráfego MPI isolado sobre a plataforma SDN proposta. Os experimentos seguem uma metodologia específica de avaliação e os resultados obtidos indicam a estratégia de roteamento mais apropriada a esse tipo de aplicação, levando em conta o ambiente de testes. A estratégia de roteamento aqui definida é posteriormente utilizada na avaliação de aplicações reais da Seção 5.5.

### 5.3.1 **Metodologia de Avaliação**

Nesta avaliação, montamos o ambiente de experimentação em um servidor 64 bits com duas CPUs Intel Xeon E5520 2,27 GHz, 16 núcleos, memória RAM de 48 GB, HD de 1 TB e sistema operacional Ubuntu 14.10. Preparamos o modelo de testes visando avaliar os quatro algoritmos previamente propostos. Executamos um *benchmark* MPI em paralelo a partir do *host* $h1$ utilizando nós de trabalho distribuídos por diversos PODs. Nesse caso, empregamos a ferramenta HPCC (HPC *Challenge Benchmark*) versão 1.5.0 (HPCC,

2019). A *suite* HPCC consiste de basicamente sete testes que incluem diferentes métricas de desempenho. Entre eles, focamos em um conjunto de testes baseado no b_eff (*effective bandwidth benchmark*) (B_EFF, 2019), o qual mede a vazão e a latência de vários padrões de comunicação simultâneos. Para tanto, o b_eff usa diversos métodos e tamanhos de mensagens. Seu algoritmo utiliza uma média que considera que aplicações reais transferem mensagens curtas e longas com diferentes valores de vazão.

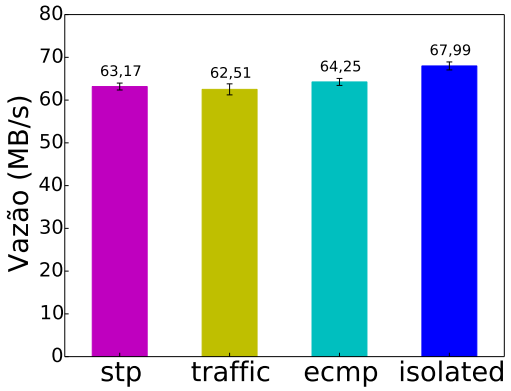
Para o projeto das simulações, definimos duas variáveis de resposta, ambas medidas sob o ponto de vista do *host* $h1$: (i) vazão utilizada na comunicação; e (ii) latência dos pacotes. Tais métricas são importantes no âmbito das redes de ambientes HPC, as quais exigem interconexões de alta vazão e baixa latência. Para as simulações paralelas e distribuídas, utilizamos três tamanhos de grupos de *hosts*: (a) 4 nós ($h1$, $h3$, $h5$ e $h7$ – PODs 1 e 2); (b) 8 nós (*hosts* ímpares – todos os PODs); e (c) 16 nós (todos os *hosts*). A partir desse planejamento, efetuamos três sessões de testes, onde para cada sessão somente um grupo de nós foi usado. Executamos cada configuração aplicando os quatro algoritmos propostos: (1) "stp"; (2) "traffic"; (3) "ecmp"; e (4) "isolated". Empregamos as configurações padrão do HPCC: medições de vazão com mensagens de 2000000 *bytes*; medições de latência com mensagens de 8 *bytes*; e comunicação lógica por anel realizada em ambas as direções. Com o intuito de determinar o impacto do erro experimental, em cada rodada de testes efetuamos 30 execuções do HPCC, com intervalos de 30 segundos entre elas. Os resultados são médias obtidas com intervalos de confiança de 95%.

### 5.3.2 **Resultados**

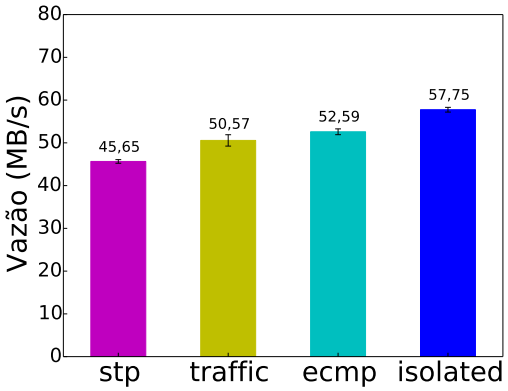
A Figura 11 exibe, para cada grupo de nós, a vazão média e os respectivos intervalos de confiança em função do algoritmo escolhido. Podemos notar que há uma clara influência do processo de seleção de caminhos no desempenho da comunicação MPI, para todos os cenários avaliados. Essa influência é menos acentuada na sessão de testes com 4 nós (Figura 11a), com os algoritmos "stp" e "traffic" apresentando, inclusive, taxas equivalentes de vazão (63,17 MB/s e 62,51 MB/s, respectivamente). Os algoritmos "ecmp", com 64,25 MB/s, e "isolated", com 67,99 MB/s, alcançaram melhores resultados nessa sessão. O ganho do melhor algoritmo em relação ao pior foi de cerca de 9%, o que sugere uma correlação direta entre o número de mensagens na rede e os possíveis benefícios do uso de estratégias de roteamento distintas.

De fato, os gráficos das sessões de testes com 8 e 16 nós mostram mais claramente a relevância da definição de um método de roteamento apropriado na melhoria do desempenho da rede. Como indica a Figura 11b, na sessão de testes com 8 nós, o ganho relativo do "isolated" (vazão de 57,75 MB/s) em comparação com o "stp" (vazão de 45,65 MB/s) foi superior a 26%. Ainda, os algoritmos "traffic", com 50,57 MB/s, e "ecmp", com 52,59 MB/s, também apresentaram melhores taxas de vazão em relação ao "stp". A sessão de testes

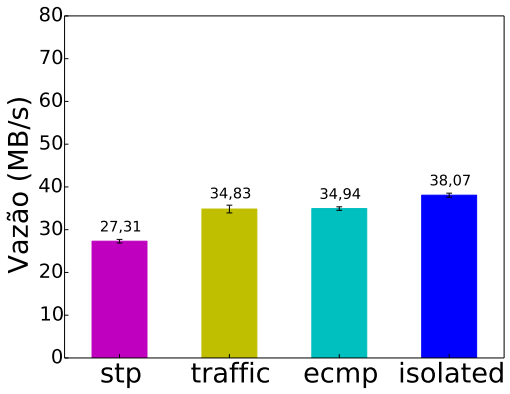
com 16 nós (Figura 11c), embora aponte menores médias, revelou diferenças expressivas de taxas de vazão entre os algoritmos “stp” (27,31 MB/s) e “isolated” (38,07 MB/s), o que representa uma melhoria de aproximadamente 39%. Nesse mesmo cenário, os algoritmos “traffic” e “ecmp” também proporcionaram melhores desempenhos, com taxas de vazão de 34,83 MB/s e 34,94 MB/s, respectivamente.




(a) 4 Nós



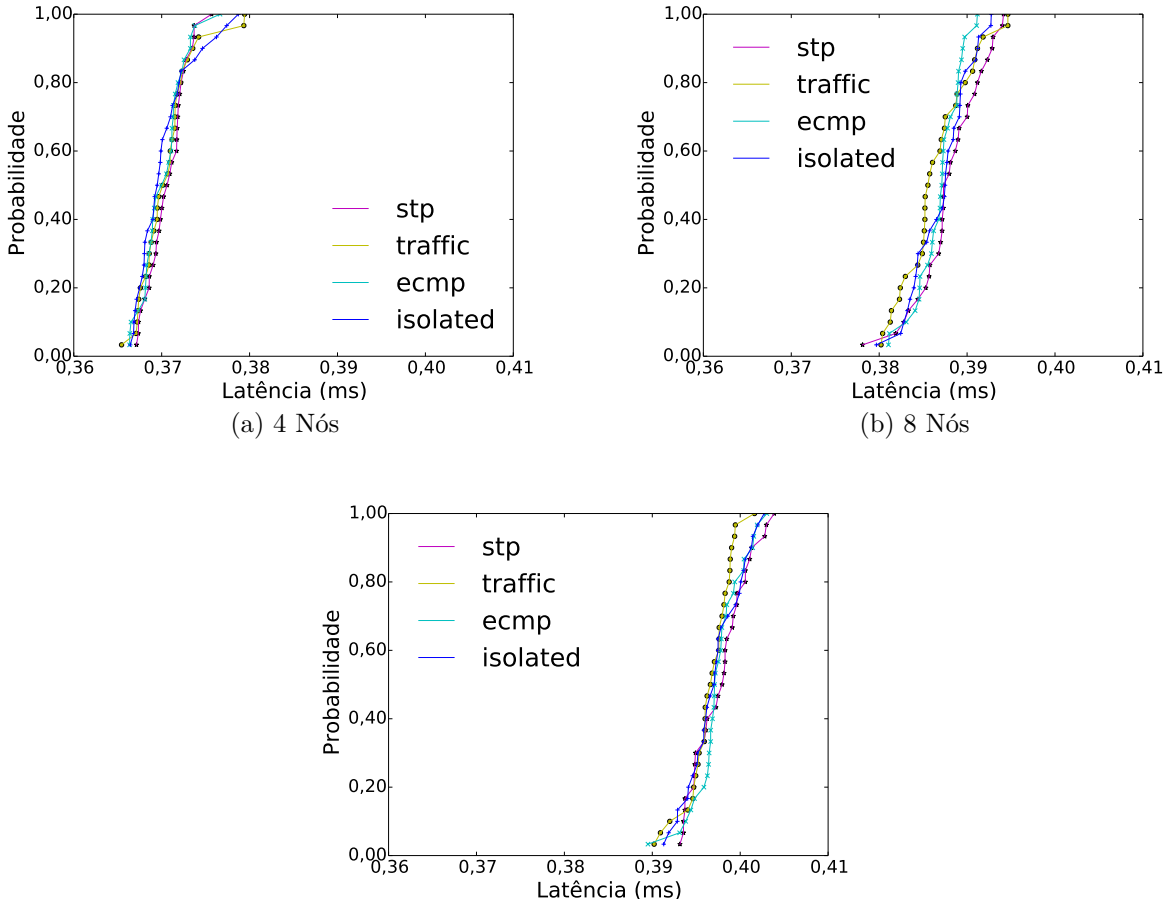

(b) 8 Nós




(c) 16 Nós

Figura 11 – Médias das taxas de vazão na comunicação MPI.

Nesta avaliação, fomos capazes de analisar a latência fim-a-fim dos pacotes em todas as sessões de testes, haja vista que o *benchmark* HPCC afere nativamente essa métrica. A Figura 12 mostra, para cada conjunto de nós, as respectivas funções de distribuição de probabilidade acumulada em relação aos algoritmos utilizados. Os gráficos demonstram que o desempenho da rede permaneceu estável no que se refere a esse parâmetro, mesmo diante do uso de métodos de roteamento distintos. Realmente, os valores médios apresentaram-se muito próximos entre si: 0,37 ms (4 nós – Figura 12a); 0,39 ms (8 nós – Figura 12b); e 0,40 ms (16 nós – Figura 12c). Os resultados também evidenciam que não houve contenção de rede no cenário.




(a) 4 Nós

(b) 8 Nós

(c) 16 Nós

Figura 12 – Distribuição de latência dos pacotes na comunicação MPI.

### 5.3.3 **Sumarização**

O algoritmo de roteamento “isolated”, que utiliza a estratégia de isolamento lógico de caminhos, apresentou o melhor desempenho de rede em todos os cenários da avaliação com o tráfego HPC. O resultado mais expressivo foi obtido na sessão de testes onde o *benchmark* MPI foi executado em paralelo utilizando todos os 16 nós de trabalho do cenário. Nessa sessão, o emprego do algoritmo “isolated” aumentou a vazão utilizada na comunicação em aproximadamente 39% em relação ao algoritmo “stp”. A latência fim-a-fim dos pacotes permaneceu estável em todas as sessões de testes, com valores máximos em torno de 0,40 ms. Essas medições indicam que não houve contenção de rede durante os experimentos. Diante desses resultados, constatamos a nítida influência do processo de seleção de rotas no desempenho da comunicação, principalmente quando há um grande número de mensagens na rede.

## 5.4 AVALIAÇÃO DE REDE COM TRÁFEGO BIG DATA

Seguindo os mesmos critérios gerais da avaliação com tráfego HPC da Seção 5.3, buscamos definir a configuração de rede mais adequada às aplicações típicas de *Big Data*. Para tal, de forma isolada, realizamos a análise de um tráfego MapReduce Hadoop sobre a plataforma SDN proposta. A metodologia de avaliação aplicada e os resultados obtidos a partir das experimentações indicam a melhor estratégia de roteamento para essas aplicações, considerando o cenário de execução. Da mesma forma que na avaliação anterior, essa estratégia de roteamento também é posteriormente usada na avaliação com aplicações reais da Seção 5.5.

### 5.4.1 **Metodologia de Avaliação**

Nesta avaliação, montamos o ambiente de simulação sobre o mesmo sistema computacional utilizado na avaliação com tráfego HPC (vide Subseção 5.3.1). Preparamos o modelo de testes em um ambiente de execução de aplicações MapReduce, procurando avaliar os mesmos quatro algoritmos de roteamento propostos anteriormente. Para os experimentos, utilizamos a ferramenta MRemu (NEVES et al., 2015), um *framework* baseado em emulação para pesquisa em rede usando cargas de trabalho MapReduce. MRemu usa traços de rede, criados a partir de execuções de aplicações Hadoop em *clusters* reais, como base para produzir cargas de tráfego emuladas no Mininet. Internamente, essas cargas são geradas pela ferramenta iPerf (IPERF, 2019). MRemu abstrai o processamento nos nós de trabalho do *cluster*, focando no tráfego de rede. Assim, o ambiente emulado não necessita de grande capacidade de processamento.

Para o projeto das simulações, selecionamos três traços de rede, os quais são distribuídos juntamente com o MRemu a partir da página do projeto na plataforma de hospedagem de código-fonte GitHub (NEVES, 2019). Esses traços variam em função do número de tarefas *map*/*reduce*, produzindo quantidades distintas de transferências na rede: (a) 128x4 (128 *maps*/4 *reduces* – 512 transferências); (b) 192x16 (192 *maps*/16 *reduces* – 3072 transferências); e (c) 256x16 (256 *maps*/16 *reduces* – 4096 transferências). Os traços usados geram tráfego entre os 16 *hosts* do cenário. A partir desse planejamento, executamos três sessões de testes, onde para cada sessão somente um traço de rede foi usado. Cada configuração foi executada utilizando os algoritmos previamente descritos: (1) “stp”; (2) “traffic”; (3) “ecmp”; e (4) “isolated”. Em cada rodada de testes efetuamos 30 execuções do MRemu a partir do *host* $h1$, com intervalos de 30 segundos entre elas. Definimos o tempo de execução do emulador como variável de resposta única, tendo em vista que o MRemu calcula, de forma nativa, somente essa métrica. Diante disso, dispensamos a avaliação de índices como vazão e latência, considerando que a implementação de mecanismos próprios de aferição poderia apresentar medições imprecisas no ambiente emulado. Os resultados dos tempos de execução são médias obtidas com intervalos de confiança de 95%.

### 5.4.2 **Resultados**

A Figura 13 apresenta, para cada traço de rede, o tempo de execução do emulador de aplicações Hadoop e os respectivos intervalos de confiança em função do algoritmo escolhido. Nota-se uma influência do processo de seleção de caminhos no desempenho da comunicação de dados, para todas as medições. Na sessão de testes com o traço 128x4 (Figura 13a) essa influência é relativa, com os algoritmos "traffic", "ecmp" e "isolated" apresentando tempos de execução bem próximos entre si (157,52 s, 156,10 s e 157,02 s, respectivamente). Em relação ao algoritmo "stp", com tempo de execução de 188,69 s, esses algoritmos alcançaram resultados significativamente melhores. De fato, o ganho do melhor algoritmo ("ecmp") em relação ao pior ("stp") foi de mais de 17%.

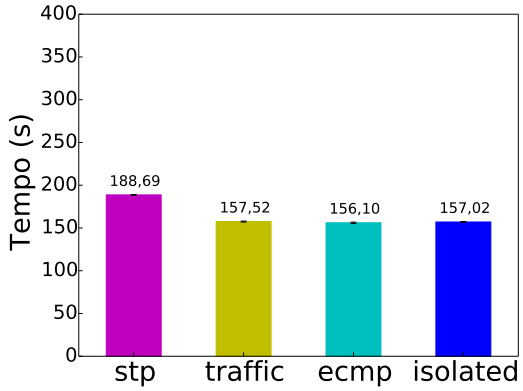
As Figuras 13b e 13c mostram claramente a relevância da definição de um método de roteamento apropriado no desempenho da rede. Como indica a Figura 13b, na sessão de testes com o traço 192x16, o ganho do "ecmp" (tempo de 137,71 s) em comparação com o "stp" (tempo de 177,49 s) foi superior a 22%. Nessa sessão, o algoritmo "ecmp" obteve tempos de execução 10% menores em relação ao algoritmo "traffic" (tempo de 153,02 s) e quase 3% menores em comparação com o algoritmo "isolated" (tempo de 141,27 s). Por sua vez, a sessão de testes com o traço 256x16 (Figura 13c) também revelou diferenças relevantes de tempos de execução entre os algoritmos "ecmp" (294,33 s) e "stp" (335,30 s), o que representa uma melhoria acima de 12%. Nesses mesmos testes, os algoritmos "traffic" e "isolated" também proporcionaram melhores desempenhos, com tempos de execução de 304,79 s e 297,28 s, respectivamente.

### 5.4.3 **Sumarização**

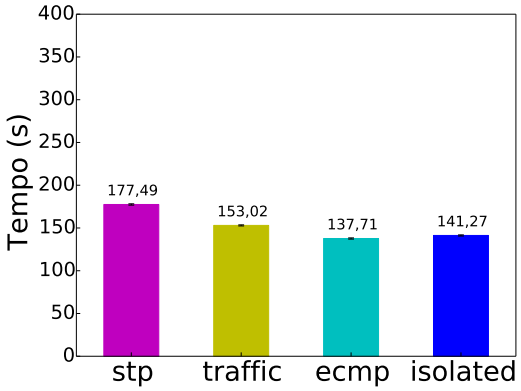
Para o cenário de avaliação descrito nesta Seção 5.4, o algoritmo de roteamento "ecmp", que utiliza a estratégia de seleção de caminhos por meio de funções de *hash*, apresentou o melhor desempenho em todas as sessões de testes. O resultado mais significativo foi alcançado na sessão de testes com o traço de rede gerando 192 tarefas *map* e 16 tarefas *reduce*. Nesse caso, o algoritmo "ecmp" reduziu o tempo de execução da emulação em mais de 22% quando comparado ao "stp". De forma análoga à avaliação de rede com tráfego HPC, a experimentação com tráfego *Big Data* também evidenciou a relevância da definição de uma política de roteamento apropriada, embora o ganho de desempenho pareça ser mais dependente do ambiente de avaliação como um todo.

## 5.5 AVALIAÇÃO DE DESEMPENHO DE APLICAÇÕES REAIS

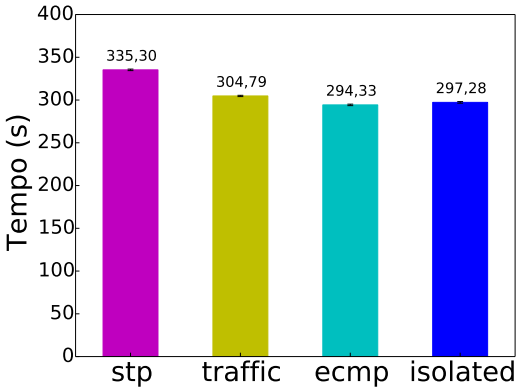
Nesta seção, analisamos o desempenho de aplicações reais de computação paralela e distribuída sobre a plataforma SDN, considerando o cenário exibido na Seção 5.1 e os critérios gerais citados na Seção 5.2. Para tanto, avaliamos o uso de aplicações Hadoop e MPI localizadas conjuntamente no mesmo ambiente de testes, utilizando a




(a) Traço 128x4




(b) Traço 192x16




(c) Traço 256x16

Figura 13 – Médias dos tempos de execução do emulador de aplicações Hadoop.

API fornecida pela plataforma e aplicando as estratégias de roteamento indicadas nas avaliações das Seções 5.3 e 5.4. A metodologia de avaliação e os resultados obtidos a partir das configurações de rede apropriadas mostram a efetividade de uso da plataforma SDN proposta.

### 5.5.1 Configurações de Rede

Nesta análise, configuramos na plataforma SDN mecanismos de identificação de tráfego proativos baseados na API apresentada no Capítulo 4. Para isso, desenvolvemos uma simples aplicação administrativa de teste. Associamos o tipo de aplicação '*hadoop*' às portas 13562, 9000, 50010 e 50020, e aos intervalos de portas 8030-8040 e 8480-8490, respeitando os padrões de comunicação da camada de transporte dos diversos componentes de um ambiente real Hadoop. Associamos também o tipo de aplicação '*mpi*' à porta 22 (SSH) e ao intervalo de portas 1024-1044 (padrão do parâmetro *btl_tcp_port_min_v4* do Open MPI). De acordo com análises prévias durante nossos testes preliminares, essas configurações permitem ao controlador SDN reconhecer mais de 86% dos fluxos de pacotes gerados por uma aplicação Hadoop e quase 100% dos fluxos gerados por uma aplicação

Open MPI. As diferenças para a totalidade referem-se a fluxos de pacotes que usam portas dinâmicas aleatórias, impossibilitando a sua associação.

Aplicamos à plataforma de rede os algoritmos de roteamento mais adequados a esse cenário de testes, tomando como base nossas avaliações anteriores. Sendo assim, mapeamos a aplicação MPI ao algoritmo "isolated", conforme análise da Seção 5.3. Por sua vez, para a aplicação Hadoop, aplicamos o algoritmo "ecmp", tendo em vista a avaliação da Seção 5.4. Para os fluxos de pacotes de dados não associados a nenhuma das aplicações, configuramos um algoritmo de roteamento comum e que minimiza o custo computacional da plataforma. Nesse caso, aplicamos o "stp", que comporta-se da mesma forma que o *Spanning Tree Protocol* e suas variações, os quais são habilitados por padrão na imensa maioria dos comutadores de rede utilizados em infraestruturas de comunicação corporativas/profissionais.

### 5.5.2 **Metodologia de Avaliação**

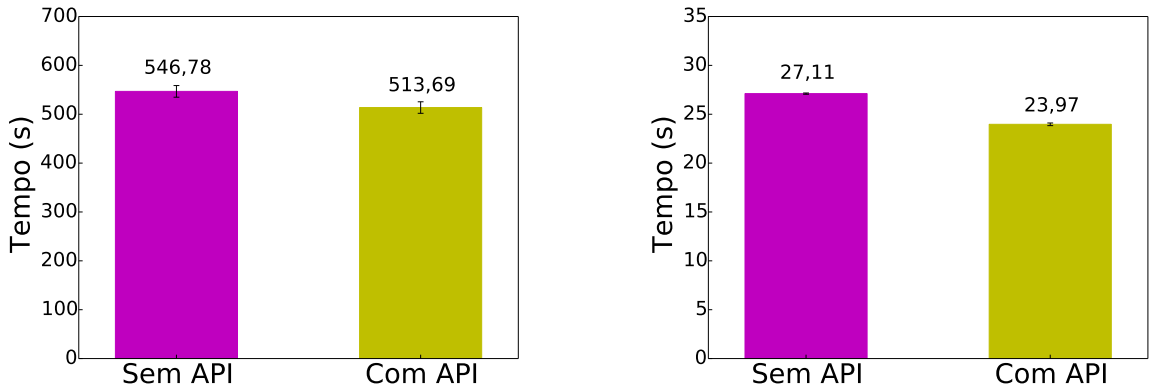
Nesta análise de aplicações reais, consideramos a mesma arquitetura SDN utilizada nas demais avaliações. No entanto, para a simulação do ambiente, utilizamos o Containernet (PEUSTER et al., 2016), um *fork* do Mininet que permite usar contêineres Docker (DOCKER, 2019) como *hosts* em redes emuladas. Containernet supera uma restrição do Hadoop diante da qual ele exige que cada nó de trabalho do *cluster* tenha um *hostname* separado, o que o impossibilita de ser executado nos *hosts* virtuais do Mininet (QIAO et al., 2016). Assim, usamos contêineres Docker com as devidas distribuições e configurações para formar um *cluster* Hadoop. Devido às maiores exigências computacionais desse ambiente, executamos a experimentação em um servidor mais robusto, com duas CPUs Intel Xeon E5-2620 2,40 GHz 64 bits, 24 núcleos, memória RAM de 64 GB, HD de 4 TB e sistema operacional Suse Linux Enterprise Server (SLES) 12.

No diagnóstico MapReduce, utilizamos o Hadoop 2.7.2 e sua aplicação de ordenação de dados Terasort. Para a geração desses dados, usamos a aplicação Teragen. Essas aplicações são empacotadas e fornecidas juntamente com a distribuição Hadoop. Com Terasort/Teragen é possível avaliar o desempenho geral do Hadoop, pois ele combina as capacidades de distribuição (HDFS) e de processamento dos dados (*map*/*reduce*) no *cluster*. No ambiente Hadoop/YARN, o *host* $h1$ foi designado unicamente como *master*, atuando como *NameNode* e *ResourceManager*. Os demais *hosts* ($h2..h16$) exerceram ao mesmo tempo o papel de *DataNode* e *NodeManager*. No diagnóstico MPI, empregamos o Open MPI 1.6.5 para a execução paralela e distribuída de um código de resolução numérica de equação de calor em três dimensões (Heat 3D). Utilizamos uma versão paralelizada por MPI em linguagem C dessa aplicação, a qual está disponível em DOURNAC (2019). Essa aplicação caracteriza-se por um paralelismo de granularidade fina, que incrementa a comunicação de dados na rede.

Para o projeto das simulações, estabelecemos as análises das duas aplicações considerando as seguintes condições: (a) Hadoop Terasort com conjunto de dados de 10 GB; e (b) MPI Heat 3D com matriz de 32x64x16. Nesses termos, ambas as aplicações geram um intenso tráfego de dados na rede, o que permite avaliar a plataforma em um contexto adequado. Para fins comparativos, executamos duas sessões de testes, onde em cada sessão a plataforma opera de duas formas diferentes: (1) Sem API – controlador SDN ignora as informações configuradas e seleciona os caminhos aplicando um algoritmo de roteamento genérico; e (2) Com API – controlador SDN identifica a aplicação através da API e aplica o roteamento mais adequado. Na sessão sem API, o algoritmo utilizado foi o "stp". Em cada rodada de testes efetuamos 15 execuções de cada aplicação a partir do *host* $h1$, com intervalos de 30 segundos entre elas. Definimos o tempo de execução como variável de resposta. Os resultados médios possuem níveis de confiança de 95%.

### 5.5.3 **Resultados**

A Figura 14 exibe os resultados obtidos nos testes das aplicações Hadoop Terasort e MPI Heat 3D, considerando as duas formas de operação da plataforma (sem API e com API). Apesar da sobrecarga adicionada pelos mecanismos de identificação de tráfego, constatamos que o uso da API assegurou uma diminuição nos tempos médios de execução dessas aplicações. Como aponta a Figura 14a, na sessão de testes com Hadoop, a plataforma operando com API completou a execução da aplicação em um tempo médio de 513,69 s, o que representa uma redução em torno de 6% em comparação com o cenário sem API, que a executou em um tempo médio de 546,78 s. Na sessão de testes com a aplicação MPI (Figura 14b), a melhoria de desempenho foi mais significativa. Nessa sessão, a execução sobre a plataforma com API (23,97 s) foi acima de 11% mais rápida em relação à plataforma sem API (27,11 s).




(a) Hadoop Terasort

(b) MPI Heat 3D

Figura 14 – Médias dos tempos de execução das aplicações reais Hadoop e MPI.

### 5.5.4 **Sumarização**

As diminuições dos tempos de execução das aplicações reais avaliadas nesta seção comprovam a viabilidade e a aplicabilidade da plataforma SDN proposta. Certamente, os índices obtidos —redução acima de 11% para a aplicação MPI e cerca de 6% para a aplicação Hadoop— representam melhorias relevantes, principalmente quando se leva em conta as exigências de alto desempenho dos ambientes de computação paralela e distribuída. Por fim, os resultados validam nossa abordagem de otimização da comunicação por mecanismos de roteamento dinâmico.

# 6 TRABALHOS RELACIONADOS

A necessidade sempre crescente por capacidade de processamento computacional continuamente motivou trabalhos de pesquisa no campo da programação paralela e da computação de alto desempenho. Com o amadurecimento do paradigma SDN, o estudo das redes de dados nos ambientes de computação avançada ganhou novos contornos, especialmente na otimização das aplicações típicas de HPC e dos *frameworks Big Data*. Nesse aspecto, muitas dessas pesquisas se relacionam de alguma forma com a proposta e o objetivo desta dissertação.

Na literatura, existem trabalhos que empregam SDN para aprimorar o processamento MPI. Por exemplo, TAKAHASHI et al. (2018) apresentam um mecanismo de coordenação de controle de rede e execução de aplicações definido por *software*. Esse mecanismo é baseado em um *framework* genérico MPI aprimorado por SDN, o qual incorpora o controle flexível de rede do modelo SDN em uma biblioteca MPI. No trabalho de BHATIA et al. (2017), o controlador SDN utiliza estatísticas de fluxo para atualizar um grafo de topologia de rede usado por um algoritmo de roteamento adaptativo na seleção dinâmica dos caminhos. Por sua vez, ALSMADI et al. (2016) usam informações de rede, obtidas pelo controlador, e empregam um orquestrador de recursos para selecionar os nós mais adequados ao processamento de cada tarefa.

Há ainda na literatura algumas pesquisas que utilizam SDN na melhoria do processamento de dados em larga escala do Hadoop. Por exemplo, os trabalhos de LIANG; LAU (2016) e NARAYAN et al. (2012) exploram o uso de SDN para maximizar a utilização da largura de banda da rede durante a fase de embaralhamento (*shuffle*) dos *jobs* MapReduce. No primeiro estudo, os autores apresentam BAShuffler, um agendador de recursos consciente da largura de banda que seleciona os nós de origem mais apropriados à otimização da rede. Implementado e embarcado no gerenciador de *cluster* YARN, BAShuffler usa um método que estima a utilização de largura de banda considerando a comunicação TCP. No segundo trabalho, os autores adotam uma abordagem na qual filas de QoS com taxas diferenciadas são configuradas na rede. O objetivo é decrementar o tempo que *Reducers* têm que esperar para reunir dados dos *Mappers*. Como outro exemplo, QIN et al. (2015) também propõem um agendador de tarefas ciente da largura de banda utilizando o modelo SDN. Nessa abordagem, chamada BASS, o agendador obtém uma avaliação completa da largura de banda da rede através do controlador SDN e a considera como um parâmetro vital para o agendamento da tarefa.

A integração entre HPC e *Big Data* é um tópico atual de pesquisa, sendo discutida e analisada sob várias perspectivas, sobretudo teóricas, em artigos como os de REED; DONGARRA (2015), FOX et al. (2015) e ASCH et al. (2018). Em parte, essa confluência é tratada de forma prática por trabalhos como o de PONCE et al. (2018), que apresentam

uma extensão do modelo de programação paralela e distribuída COMP Superscalar (COMPSs) para o processamento de dados massivos. Nesse estudo, o COMPSs é integrado ao sistema de arquivos distribuídos HDFS. Há também trabalhos cujos princípios fornecem diversas percepções em torno dessa integração em camadas mais baixas no contexto de redes, embora não abordem esse tema de forma explícita. Por exemplo, WEBB et al. (2011) formalizam a possibilidade de uso simultâneo de múltiplos mecanismos de roteamento em um *data center*, permitindo às aplicações defini-los e implantá-los conforme suas necessidades.

De forma geral, os trabalhos que exploram as capacidades do paradigma SDN nos ambientes HPC e *Big Data* o fazem de maneira isolada, como evidenciam os artigos de TAKAHASHI et al. (2018), BHATIA et al. (2017) e ALSMADI et al. (2016), que tratam da melhoria do processamento MPI, e os estudos de LIANG; LAU (2016), NARAYAN et al. (2012) e QIN et al. (2015), que abordam a otimização do processamento Hadoop. Nossa proposta de plataforma, por sua vez, tira proveito da flexibilidade proporcionada por SDN para prover uma infraestrutura de comunicação verdadeiramente integrada e, acima de tudo, eficiente. Nesse aspecto, embora PONCE et al. (2018) dediquem-se ao tema da integração, eles focam no nível da aplicação.

Nossa pesquisa também baseia-se em um modelo onde a rede é ciente da aplicação, com a rede ajustando-se aos requisitos das aplicações, em contraponto aos estudos de ALSMADI et al. (2016), LIANG; LAU (2016) e QIN et al. (2015), que propõem soluções sob um ponto de vista no qual a aplicação se torna consciente da rede, ou seja, adapta-se às condições da rede subjacente sem alterá-la. Por fim, no contexto dos métodos de otimização da comunicação, nossa proposta assemelha-se à de WEBB et al. (2011), ao fazer uso de mecanismos de roteamento distintos, enquanto que NARAYAN et al. (2012) recorrem a conceitos de QoS para maximizar o tráfego de dados.

A Tabela 1 sumariza os trabalhos relacionados descritos neste capítulo, associando-os aos ambientes típicos de HPC ou de *Big Data*, aos ambientes integrados ou aos ambientes de *data center*. Para os estudos que tratam de ambientes integrados, essa tabela também vincula os aspectos de atuação de cada um deles, classificando-os de acordo com uma perspectiva de análise teórica, de enfoque na camada de aplicação ou de foco na camada de rede. A Tabela 1 ainda associa os trabalhos que exploram as capacidades de SDN aos modelos de aplicação consciente da rede ou de rede ciente da aplicação, correlacionando esses últimos, por fim, aos respectivos mecanismos de otimização de rede, quer seja por roteamento ou por provisão de QoS. Ao final do quadro, com o propósito de traçar um paralelo com os demais trabalhos, nossa proposta de plataforma SDN é categorizada de acordo com os atributos da Tabela 1.

Tabela 1 – Sumário dos trabalhos relacionados.

| | Ambiente | | | | Aspectos da Integração | | | Abordagem | | Mecanismo de Otimização | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | HPC | Big Data | Integrado | Data Center | Teórico | Aplicação | Rede | Aplicação Consciente da Rede | Rede Ciente da Aplicação | Roteamento | QoS |
| TAKAHASHI et al. (2018) | x | | | | | | | | x | x | |
| BHATIA et al. (2017) | x | | | | | | | | x | x | |
| ALSMADI et al. (2016) | x | | | | | | | x | | | |
| LIANG; LAU (2016) | | x | | | | | | x | | | |
| NARAYAN et al. (2012) | | x | | | | | | | x | | x |
| QIN et al. (2015) | | x | | | | | | x | | | |
| REED; DONGARRA (2015) | | | x | | x | | | | | | |
| FOX et al. (2015) | | | x | | x | | | | | | |
| ASCH et al. (2018) | | | x | | x | | | | | | |
| PONCE et al. (2018) | | | x | | | x | | | | | |
| WEBB et al. (2011) | | | | x | | | | | | x | |
| Plataforma SDN | | | x | | | | x | | x | x | |

# 7 CONCLUSÕES

Esta dissertação apresentou uma proposta de plataforma de rede definida por *software* (SDN) cujo propósito é suprir, de forma integrada, os requisitos de desempenho das aplicações típicas de ambientes *Big Data* e HPC. A plataforma atende esses requisitos maximizando a comunicação dos dados na rede. Para tal, é oferecida uma API que permite a identificação do tráfego e o emprego das melhores estratégias de roteamento para cada aplicação. Diante disso, evidenciamos a importância da definição das configurações de rede mais adequadas a cada perfil de tráfego em determinado ambiente. Simulamos a arquitetura proposta em um cenário específico e examinamos tráfegos MPI e Hadoop através de ferramentas de *benchmark* e emulação. A partir dessas análises, simulamos aplicações reais e avaliamos a viabilidade e a aplicabilidade da plataforma proposta. Os resultados mostram que a API permite o ajuste dinâmico da infraestrutura de rede, tornando-a mais eficiente e contribuindo para a diminuição dos tempos de execução dessas aplicações de computação paralela e distribuída.

De fato, os ganhos que a plataforma SDN é capaz de proporcionar são intimamente ligados ao ambiente como um todo. Nesse sentido, cabe ao administrador da rede estabelecer os métodos de roteamento mais apropriados a cada cenário, considerando as aplicações em execução (conjuntos de dados, modelos de programação, granularidades de paralelismo, etc), a arquitetura da rede (topologias, tecnologias de comunicação, dispositivos de encaminhamento, etc), as capacidades de processamento (nós de trabalho, sistemas de arquivos, gerenciadores de recursos, etc), entre diversos outros critérios. Uma estratégia válida para lidar com toda essa diversidade de aspectos seria implantar um arcabouço de testes para medir o desempenho das aplicações, considerando, para isso, a adoção de implementações variadas e distintas de algoritmos de roteamento. Ainda dentro desse escopo, vale ressaltar também a relevância do uso de técnicas de otimização e da aplicação das melhores práticas no desenvolvimento dos sistemas de *software* da plataforma, além da correta seleção dos equipamentos usados na montagem da arquitetura SDN, avaliando as questões relativas ao *hardware* dos controladores SDN e à organização desses dispositivos junto à rede.

Independentemente de todas essas considerações, concluímos, sobretudo, que a exploração das capacidades de SDN no âmbito da solução proposta oferece as funcionalidades necessárias à flexibilização da infraestrutura de rede, potencializando o tráfego de dados e propiciando, de forma efetiva, um passo importante no caminho para a tão esperada integração dos ambientes de computação paralela e distribuída.

## 7.1 TRABALHOS FUTUROS

Como trabalhos futuros, pretendemos desenvolver mecanismos inteligentes de identificação e classificação de tráfego de rede para ambas as aplicações. Esperamos também analisar o comportamento da plataforma diante de tipos específicos de aplicações intensivas de dados, como por exemplo aquelas relacionadas ao processamento de grafos. Planeja-se, ainda, avaliar a utilização de ferramentas de "fatiamento" de rede como solução complementar à implementação da plataforma proposta. Por fim, almejamos medir o desempenho das aplicações paralelas e distribuídas no contexto de uma rede real, utilizando dispositivos físicos (não emulados), apesar das dificuldades inerentes a essas experimentações.

# REFERÊNCIAS

ALSMADI, I.; KHAMAISEH, S.; XU, D. Network parallelization in HPC clusters. In: **Proc. of the IEEE CSCI**, 2016.

APACHE. **The Apache Software Foundation**. 2019. `https://www.apache.org`. Acesso em: Janeiro, 2019.

ARAUJO, A.; SAMPAIO, L.; ZIVIANI, A. BEEP: Balancing energy, redundancy, and performance in fat-tree data center networks. **IEEE Internet Computing**, IEEE, v. 21, n. 4, p. 44–53, 2017.

ASCH, M.; MOORE, T.; BADIA, R.; BECK, M.; BECKMAN, P.; BIDOT, T.; BODIN, F.; CAPPELLO, F.; CHOUDHARY, A.; SUPINSKI, B. de et al. Big Data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. **The International Journal of High Performance Computing Applications**, SAGE Publications Sage UK: London, England, v. 32, n. 4, p. 435–479, 2018.

B_EFF. **Effective Bandwidth (b_eff) Benchmark**. 2019. `https://fs.hlrs.de/projects/par/mpi//b_eff/`. Acesso em: Janeiro, 2019.

BHATIA, S.; SINHA, Y.; CHALAPATHI, G.; KUMAR, R. MPI aware routing using SDN. **Poster Presented at the 26th International Symposium on High Performance Parallel and Distributed Computing (HPDC)**, 2017.

CASADO, M.; FREEDMAN, M.; PETTIT, J.; LUO, J.; MCKEOWN, N.; SHENKER, S. Ethane: Taking control of the enterprise. In: ACM. **ACM SIGCOMM Computer Communication Review**, 2007. v. 37, n. 4, p. 1–12.

COMER, D. **Interligação de Redes com TCP/IP: Princípios, protocolos e arquitetura**. 6. ed., 2015. Elsevier Brasil.

COSTA, L.; VIEIRA, A.; MACEDO, D. **Balanceamento de carga utilizando planos de dados OpenFlow comerciais**. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas, Junho 2016.

COSTA, L.; VIEIRA, A.; SILVA e E.; MACEDO, D.; GOMES, G.; CORREIA, L.; VIEIRA, L. Performance evaluation of OpenFlow data planes. In: IEEE. **Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on**, 2017. p. 470–475.

DATE, S.; ABE, H.; KHURELTULGA, D.; TAKAHASHI, K.; KIDO, Y.; WATASHIBA, Y.; U-CHUPALA, P.; ICHIKAWA, K.; YAMANAKA, H.; KAWAI, E.; SHIMOJO, S.

SDN-accelerated HPC infrastructure for scientific research. **International Journal of Information Technology**, v. 22, n. 1, 2016.

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

DIJKSTRA, E. A note on two problems in connexion with graphs. **Numerische mathematik**, Springer, v. 1, n. 1, p. 269–271, 1959.

DOCKER. **Enterprise container platform**. 2019. `https://www.docker.com`. Acesso em: Janeiro, 2019.

DORIA, A.; SALIM, J.; HAAS, R.; KHOSRAVI, H.; WANG, W.; DONG, L.; GOPAL, R.; HALPERN, J. **Forwarding and Control Element Separation (ForCES) Protocol Specification**, March 2010. Internet Engineering Task Force, RFC5810. Disponível em: <https://tools.ietf.org/html/rfc5810>.

DORIA, A.; SUNDELL, K. **General Switch Management Protocol (GSMP) Applicability**, June 2002. Internet Engineering Task Force, RFC3294. Disponível em: <https://tools.ietf.org/html/rfc3294>.

DOURNAC, F. **MPI Parallelization for numerically solving the 3D Heat equation**. 2019. `https://dournac.org/info/parallel_heat3d`. Acesso em: Janeiro, 2019.

ECONOMIST. The data deluge. **Special Supplement**, 2010.

FLINK. **Stateful Computations over Data Streams**. 2019. `https://flink.apache.org`. Acesso em: Janeiro, 2019.

FLOODLIGHT. **Project Floodlight**. 2019. `http://www.projectfloodlight.org`. Acesso em: Janeiro, 2019.

FLYNN, M. Some computer organizations and their effectiveness. **IEEE transactions on computers**, IEEE, v. 100, n. 9, p. 948–960, 1972.

FOX, G.; QIU, J.; JHA, S.; EKANAYAKE, S.; KAMBURUGAMUVE, S. Big Data, simulations and HPC convergence. In: **Big Data Benchmarking**, 2015. p. 3–17.

GUEDES, D.; VIEIRA, L.; VIEIRA, M.; RODRIGUES, H.; NUNES, R. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, v. 30, n. 4, p. 160–210, 2012.

HADOOP. **Apache Hadoop**. 2019. `https://hadoop.apache.org`. Acesso em: Janeiro, 2019.

HDFS. **HDFS Architecture Guide**. 2019. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Acesso em: Janeiro, 2019.

HPCC. **HPC Challenge Benchmark**. 2019. https://icl.utk.edu/hpcc/. Acesso em: Janeiro, 2019.

INFINIBANDTA. **InfiniBand™ Architecture Specification**. 2019. https://www.infinibandta.org/ibta-specification/. Acesso em: Janeiro, 2019.

IPERF. **iPerf - The ultimate speed test tool for TCP, UDP and SCTP**. 2019. https://iperf.fr. Acesso em: Janeiro, 2019.

KREUTZ, D.; RAMOS, F.; VERíSSIMO, P.; ROTHENBERG, C.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, 2015.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In: ACM. **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**, 2010. p. 19.

LEISERSON, C. Fat-trees: universal networks for hardware-efficient supercomputing. **IEEE Transactions on Computers**, IEEE, v. 100, n. 10, p. 892–901, 1985.

LIANG, F.; LAU, F. BAShuffler: Maximizing network bandwidth utilization in the shuffle of YARN. In: ACM. **Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing**, 2016. p. 281–284.

LIU, N.; HAIDER, A.; JIN, D.; SUN, X. Modeling and simulation of extreme-scale fat-tree networks for HPC systems and data centers. **ACM Transactions on Modeling and Computer Simulation (TOMACS)**, ACM, v. 27, n. 2, p. 13, 2017.

LNCC. **Configuração do SDumont**. 2019. https://sdumont.lncc.br/machine.php?pg=machine#. Acesso em: Janeiro, 2019.

LUSTRE. **Welcome to the official home of the Lustre filesystem**. 2019. http://lustre.org. Acesso em: Janeiro, 2019.

MACEDO, D.; GUEDES, D.; VIEIRA, L.; VIEIRA, M.; NOGUEIRA, M. Programmable networks—from software-defined radio to software-defined networking. **IEEE communications surveys & tutorials**, IEEE, v. 17, n. 2, p. 1102–1125, 2015.

MATTSON, T.; SANDERS, B.; MASSINGILL, B. **Patterns for Parallel Programming**, 2004. Pearson Education.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: Enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69–74, 2008.

MELLANOX. **SX6536 Product Brief**. 2015. `http://www.mellanox.com/related-docs/prod_ib_switch_systems/PB_SX6536.pdf`. Acesso em: Janeiro, 2019.

MELLANOX. **Unified Fabric Manager (UFM®) Software for Data Center Management**. 2019. `http://www.mellanox.com/page/products_dyn?product_family=100`. Acesso em: Janeiro, 2019.

MININET. **An Instant Virtual Network on your Laptop (or other PC)**. 2019. `http://mininet.org`. Acesso em: Janeiro, 2019.

MPI. **MPI Forum**. 2019. `https://www.mpi-forum.org`. Acesso em: Janeiro, 2019.

MPICH. **High-Performance Portable MPI**. 2019. `https://www.mpich.org`. Acesso em: Janeiro, 2019.

NARAYAN, S.; BAILEY, S.; DAGA, A. Hadoop acceleration in an OpenFlow-based cluster. In: **Proc. of the IEEE SCC**, 2012.

NEVES, M. **Hadoop MapReduce network emulator**. 2019. `https://github.com/mvneves/mremu`. Acesso em: Janeiro, 2019.

NEVES, M.; De Rose, C.; KATRINIS, K. MRemu: An emulation-based framework for datacenter network experimentation using realistic MapReduce traffic. In: IEEE. **Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on**, 2015. p. 174–177.

NUNES, B.; MENDONCA, M.; NGUYEN, X.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 3, p. 1617–1634, 2014.

OLIVEIRA, A.; MARTINS, B.; GOMES, A.; ZIVIANI, A.; MORENO, M.; VIEIRA, A. Arquitetura de provisão de qualidade de serviço para aplicações distribuídas de alto desempenho em redes definidas por software. In: SBC. **Anais do XXIII Workshop de Gerência e Operação de Redes e Serviços (WGRS-SBRC 2018)**, 2018. v. 23.

OLIVEIRA, A.; MARTINS, B.; MORENO, M.; VIEIRA, A.; GOMES, A.; ZIVIANI, A. SDN-Based architecture for providing QoS to high performance distributed applications.

In: IEEE. **2018 IEEE Symposium on Computers and Communications (ISCC)**, 2018. p. 602–607.

OLIVEIRA, A.; VIEIRA, A.; GOMES, A.; ZIVIANI, A. Análise de desempenho de rede para aplicações MPI em infraestruturas SDNs convergentes para HPC e Big Data. In: SBC. **Proc. of the XIX Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)**, 2018. p. 397–408.

ONF. **What is ONF?** 2013. `https://www.opennetworking.org/images/stories/downloads/about/onf-what-why.pdf`. Acesso em: Janeiro, 2019.

ONF. **Our Mission**. 2019. `https://www.opennetworking.org/mission/`. Acesso em: Janeiro, 2019.

OPENDAYLIGHT. **ODL Fluorine is here!** 2019. `https://www.opendaylight.org`. Acesso em: Janeiro, 2019.

OPENMP. **The OpenMP API specification for parallel programming**. 2019. `https://www.openmp.org`. Acesso em: Janeiro, 2019.

OPENMPI. **Open MPI: Open Source High Performance Computing**. 2019. `https://www.open-mpi.org`. Acesso em: Janeiro, 2019.

ORNL. **SUMMIT – Oak Ridge National Laboratory's next High Performance Supercomputer**. 2019. `https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/`. Acesso em: Janeiro, 2019.

OVS. **Production Quality, Multilayer Open Virtual Switch**. 2019. `https://www.openvswitch.org`. Acesso em: Janeiro, 2019.

PEIXOTO, T.; VIEIRA, A. **Aumentando a resiliência em SDN quando o Plano de Controle se encontra sob ataque**. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas, Agosto 2017.

PEIXOTO, T.; VIEIRA, A.; NOGUEIRA, M.; MACEDO, D. Does OpenFlow really decouple the data plane from the control plane? In: IEEE. **2018 IEEE Symposium on Computers and Communications (ISCC)**, 2018. p. 596–601.

PEUSTER, M.; KARL, H.; Van Rossem, S. MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments. **IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA**, p. 148–153, 2016.

PONCE, L.; SANTOS, W.; MEIRA-JR, W.; GUEDES, D. Extensão de um ambiente de computação de alto desempenho para o processamento de dados massivos. In: **Proc. of the SBRC**, 2018.

PORTO, F. Algoritmos e modelos de programação em Big Data, XXXVI Jornada de Atualização em Informática (JAI). In: **Proc. of the SBC Congress**, 2017.

POX. **The POX network software platform**. 2019. `https://github.com/noxrepo/pox`. Acesso em: Janeiro, 2019.

PYTHON. **Welcome to Pyhton.org**. 2019. `https://www.python.org`. Acesso em: Janeiro, 2019.

QIAO, Y.; WANG, X.; FANG, G.; LEE, B. Doopnet: An emulator for network performance analysis of Hadoop clusters using Docker and Mininet. In: IEEE. **2016 IEEE Symposium on Computers and Communication (ISCC)**, 2016. p. 784–790.

QIN, P.; DAI, B.; HUANG, B.; XU, G. Bandwidth-aware scheduling with SDN in Hadoop: A new trend for Big Data. **IEEE Systems Journal**, IEEE, 2015.

REED, D.; DONGARRA, J. Exascale computing and Big Data. **Communications of the ACM**, ACM, v. 58, n. 7, p. 56–68, 2015.

RYU. **Build SDN Agilely**. 2019. `https://osrg.github.io/ryu/`. Acesso em: Janeiro, 2019.

SLURM. **Workload Manager Overview**. 2019. `https://slurm.schedmd.com/overview.html`. Acesso em: Janeiro, 2019.

SPARK. **Lightning-fast unified analytics engine**. 2019. `http://spark.apache.org`. Acesso em: Janeiro, 2019.

TAKAHASHI, K.; DATE, S.; KHURELTULGA, D.; KIDO, Y.; YAMANAKA, H.; KAWAI, E.; SHIMOJO, S. UnisonFlow: A software-defined coordination mechanism for message-passing communication and computation. **IEEE Access**, IEEE, v. 6, p. 23372–23382, 2018.

TANENBAUM, A. **Sistemas operacionais modernos**. 2. ed., 2003. Prentice Hall.

TENNENHOUSE, D.; SMITH, J.; SINCOSKIE, W.; WETHERALL, D.; MINDEN, G. A survey of active network research. **IEEE Communications Magazine**, IEEE, v. 35, n. 1, p. 80–86, 1997.

THALER, D.; HOPPS, C. **Multipath Issues in Unicast and Multicast Next-Hop Selection**, November 2000. Internet Engineering Task Force, RFC2991. Disponível em: <https://tools.ietf.org/html/rfc2991>.

VAVILAPALLI, V.; MURTHY, A.; DOUGLAS, C.; AGARWAL, S.; KONAR, M.; EVANS, R.; GRAVES, T.; LOWE, J.; SHAH, H.; SETH, S. et al. Apache Hadoop YARN: Yet

another resource negotiator. In: ACM. **Proceedings of the 4th annual Symposium on Cloud Computing**, 2013. p. 5.

WEATHERSPOON, H. **Data Center Network Topologies: FatTree**. 2014. `https://www.cs.cornell.edu/courses/cs5413/2014fa/lectures/08-fattree.pdf`. Acesso em: Janeiro, 2019.

WEBB, K.; SNOEREN, A.; YOCUM, K. Topology switching for data center networks. **Hot-ICE**, v. 11, 2011.

ZAHARIA, M.; CHOWDHURY, M.; FRANKLIN, M.; SHENKER, S.; STOICA, I. Spark: Cluster computing with working sets. **HotCloud**, v. 10, n. 10-10, p. 95, 2010.