CLEVERTON JULIANO ALVES VICENTINI

PROVISIONAMENTO DINÂMICO DE RECURSOS VIRTUALIZADOS EM BIG DATA STREAMING

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná, como requisito parcial à obtenção do título de Doutor em Informática.

Orientador: Prof. Dr. Altair Olivo Santin

CURITIBA

2018

CLEVERTON JULIANO ALVES VICENTINI

PROVISIONAMENTO DINÂMICO DE RECURSOS VIRTUALIZADOS EM BIG DATA STREAMING

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná, como requisito parcial à obtenção do título de Doutor em Informática.

Área de Concentração: *Ciência da Computação*

Orientador: Prof. Dr. Altair Olivo Santin

CURITIBA

2018

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Edilene de Oliveira dos Santos CRB 9 / 1636

V633p 2018

Vicentini, Cleverton Juliano Alves
Provisionamento dinâmico de recursos virtualizados em big data streaming / Cleverton Juliano Alves Vicentini ; orientador, Altair Olivo Santin. -- 2018
117 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2018.
Bibliografia: f. 109-117

1. Informática. 2. Computação em nuvem. 3. Redes de computação. 4. Engenharia de software. I. Santin, Altair Olivo. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título

CDD 20. ed. – 004

Pontifícia Universidade Católica do Paraná

PUCPR

GRUPO MARISTA

ATA DE SESSÃO PÙBLICA

DEFESA DE TESE DE DOUTORADO Nº 56/2018

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA – PPGIa
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ - PUCPR

Em sessão pública realizada às 09h00 de 28 de Maio de 2018, no Auditório Guglielmo Marconi – Bloco 8 – Térreo, ocorreu a defesa da tese de doutorado intitulada **"Provisionamento Dinâmico de Recursos Virtualizados em Big Data *Streaming*"** elaborada pelo aluno **Cleverton Juliano Alves Vicentini**, como requisito parcial para a obtenção do título de **Doutor em Informática**, na área de concentração **Ciência da Computação,** perante a banca examinadora composta pelos seguintes membros:

Prof. Dr. Altair Olivo Santin (orientador) - PUCPR

Prof. Dr. Marcelo Eduardo Pellenz - PUCPR

Prof. Dr. Lisandro Zambenedetti Granville – UFRGS

Prof. Dr. Luis C. E. de Bona – UFPR

Prof. Dr. Carlos Alberto Maziero - UFPR

Após a apresentação da tese pelo aluno e correspondente arguição, a banca examinadora emitiu o seguinte parecer sobre a tese:

Membro	Parecer
Prof. Dr. Altair Olivo Santin	(X) Aprovada ( ) Reprovada
Prof. Dr. Marcelo Eduardo Pellenz	(X) Aprovada ( ) Reprovada
Prof. Dr. Lisandro Zambenedetti Granville	(X) Aprovada ( ) Reprovada
Prof. Dr. Luis C. E. de Bona	(X) Aprovada ( ) Reprovada
Prof. Dr. Carlos Alberto Maziero	(X) Aprovada ( ) Reprovada

Portanto, conforme as normas regimentais do PPGIa e da PUCPR, a tese foi considerada:

(X) **APROVADO**

(aprovação condicionada ao atendimento integral das correções e melhorias recomendadas pela banca examinadora, conforme anexo, dentro do prazo regimental)

( ) **REPROVADO**

E, para constar, lavrou-se a presente ata que vai assinada por todos os membros da banca examinadora.
Curitiba, 28 de Maio de 2018.

Prof. Dr. Altair Olivo Santin

Prof. Dr. Lisandro Zambenedetti Granville

Prof. Dr. Marcelo Eduardo Pellenz

Prof. Dr. Luis C. E. de Bona

Prof. Dr. Carlos Alberto Maziero

Agradecimentos

Primeiramente agradeço ao Grande Arquiteto do Universo, nosso Deus que permitiu de maneira profícua o desenvolvimento desta tese de doutorado.

Também agradeço à minha companheira Roberta Rafaela Sotero Costa, que esteve ao meu lado em todos os momentos dando-me força e incentivo nessa etapa de minha formação.

À minha mãe e meu pai (*in memorian*) que desde minha infância me motivaram a alçar voos altos.

Ao meu orientador Prof. Dr. Altair Olivo Santin, que através de seu entusiasmo pela pesquisa motiva os alunos a superarem suas metas. Serei eternamente grato por sua paciência e confiança em mim depositada.

Ao Instituto Federal do Paraná (IFPR), por possibilitar meu afastamento para capacitação em nível de doutorado.

À CAPES, que através de seu programa de formação doutoral docente – Prodoutoral, auxiliou financeiramente esta pesquisa.

Ao grupo de pesquisa SecPLab, liderado pelo Prof. Altair, que proporciona aos alunos semanalmente a oportunidade de amadurecer os trabalhos, com a supervisão direta e presencial do professor.

Às amizades desenvolvidas nessa caminhada, em especial ao Vilmar Abreu Junior, Eduardo Kugler Viegas e Rafael Cruz Ribeiro, amigos e companheiros de laboratório, local onde dividimos histórias, sorrisos e momentos de angustia, obrigado pessoal!

À Cheila Cristina, Jhonatan Geremias e Nicolas de Paula que, além da amizade, sempre responderam com agilidade às minhas necessidades de cunho acadêmico.

Aos professores Marcelo Eduardo Pellenz, Carlos Alberto Maziero, Lisandro Zambenedetti Granville e Luis Carlos Erpen de Bona que, mesmo com agendas repletas de compromissos, se dispuseram a participar desta etapa tão importante do meu processo formativo.

Finalmente, agradeço a todos que de maneira direta ou indireta contribuíram para que esse trabalho fosse concluído. Obrigado!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CC	*Cloud Computing*
DyLB	*Dynamic Load Balancer*
DyS	*Dynamic Scheduler*
ERP	*Elastic Resource Provisioning*
FIFO	*First In First Out*
GENI	*Global Environment for Network Innovations*
HDFS	*Hadoop Distributed File System*
IaaS	*Infrastructure as a Service*
IDC	*International Data Corporation*
MA	*Master Agent*
NIST	*National Institute of Standards and Technology*
ODL	*OpenDaylight*
OF	*Open Flow*
ONF	*Open Network Foundation*
PaaS	*Platform as a Service*
PE	*Processing Element*
S3	*Simple Storage Service*
SA	*Slave Agent*
SaaS	*Software as a Service*
SDN	*Software Defined Network*
NFV	*Network Function Virtualization*
SRTF	*Smallest Remaining Time First*
SSL	*Secure Socket Layer*

TT	*Throughput Test Topology*
VM	*Virtual Machine*
VMM	*Virtual Machine Monitor*
WC	*Word Count Topology*
WCF	*Word Count File Topology*

Resumo

A computação em nuvem oferece recursos virtuais sob demanda aos sistemas de *Big Data* para processamento de grande volume de dados em tempo real. Sistemas implantados em infraestrutura compartilhada (*multi-tenant*) de nuvem pública podem ter seu desempenho reduzido devido ao compartilhamento de hardware com outros inquilinos (*tenants*). Nesse caso, a infraestrutura de computação em nuvem não é gerenciada pelo inquilino, dificultando a detecção de tal comprometimento. Assim, com o passar do tempo a infraestrutura pode tornar-se insuficiente, devido ao aumento da demanda computacional ou da interferência *multi-tenant*. Esta tese apresenta um mecanismo de provisionamento de recursos, baseado no estado de cada máquina virtual, independente de hipervisor, que realiza o escalonamento e reescalonamento dinâmico de tarefas para frameworks de *Big Data Stream*. Além disso, o mecanismo proposto realiza o balanceamento de carga entre vários clusters usando SDN (*Software Defined Network*). O protótipo foi implementado usando o Apache Storm (*framework* de *big data*), *Helion Eucalyptus* (nuvem computacional) e *Floodlight Controller* (SDN). A avaliação do protótipo, com aplicações sob interferência *multi-tenant*, mostrou uma melhoria de desempenho de até 50,1% para tarefas *CPU-bound*, 62,3% para tarefas *disk-bound* e 43,8% para tarefas *network-bound*. Adicionalmente, o balanceador de carga apresentou uma melhoria de 22,04% na distribuição das tarefas em relação a abordagem tradicional.

Palavras-chave: Escalonamento dinâmico para Big Data. Balanceamento de Carga. Nuvem Computacional. Redes Definidas por Software. Big Data Streaming.

Abstract

Cloud computing provides on-demand virtual resources for *Big Data* systems to process in real-time large volumes of data. Systems running in shared public cloud infrastructures (multi-tenant), may have their performance decreased due to the hardware sharing with other tenants. In such context, the cloud computing infrastructure is not managed by the tenant, making it difficult to detect such problem. Thereby, the infrastructure may become unable to process the demand over time, either due to the increase of the computational demand, or due to the multi-tenant interference. This dissertation presents a hypervisor independent resource provisioning mechanism based on the state of each virtual machine, which performs the dynamic scheduling and rescheduling of tasks for Big Data stream frameworks. In addition, the proposed mechanism performs the load balancing between several clusters by the means of the SDN (Software Defined Network). The prototype was developed using Apache Storm (Big Data framework), Helion Eucalyptus (cloud computing) and Floodlight Controller (SDN). The prototype evaluation, with applications under multi-tenant interferences, showed an performance improvement of up to 50.1% for CPU-bound tasks, 62.3% for disk-bound tasks and 43.8% for network-bound tasks. Moreover, the load balancer showed an improvement of 22.04% in the tasks distribution when compared to a traditional approach.

Keywords: Dynamic scheduling for Big Data. Load Balancing. Cloud Computing. Software Defined Network. Big Data Streaming.

Capítulo 1

Introdução

1.1. Contextualização

De maneira geral, o conceito de *Big Data* é considerado quando a utilização dos meios convencionais de processamento e armazenamento tornam-se impraticáveis para o tratamento da massa de dados. O tratamento dessa massa implica no desenvolvimento de infraestruturas de sistemas distribuídos compostas por inúmeros componentes como: sistemas de arquivos distribuídos, escalonadores de demanda, suporte a processamento em lote (*batch*) ou de tempo real (*stream*), e modelos de programação que objetivam tornar operacional todo o conceito que engloba *Big Data*. Os *frameworks* de *Big Data* baseados em lote (e.g. Apache Hadoop [Apache, 2016]) são caracterizados pelo armazenamento prévio da massa de dados para o processamento da demanda computacional. Por outro lado, os *frameworks* baseados em tempo real (e.g. Apache Storm [Storm, 2016] [Toshniwal, A. et al. 2014]) não realizam o armazenamento prévio dos dados, pois nesse caso existe uma fonte responsável pela geração contínua de dados para o processamento.

Em virtude do grande volume de dados que a *Big Data* trata, torna-se necessária uma infraestrutura de *hardware* capaz de processar tal demanda. Nesse sentido, a Computação em Nuvem (*Cloud Computing*) [Mell, Grace, 2011] tipicamente é utilizada. A computação em nuvem possui uma infraestrutura robusta que fornece características como: escalabilidade, resiliência, alto desempenho e alta escalabilidade.

Tais propriedades são, em geral, providas através do modelo *multi-tenant* (multilocatário), que permite que diferentes usuários (da nuvem) compartilhem do mesmo *hardware* físico por meio de máquinas virtuais (*Virtual Machines* - VMs) que são controladas por um hipervisor. O compartilhamento de *hardware* entre vários locatários pode impactar em variações inesperadas de desempenho para os sistemas e aplicações, devido ao acesso

simultâneo aos recursos físicos disponíveis. Adicionalmente, provedores de Computação em Nuvem ainda podem instanciar um número de máquinas virtuais superior ao que a infraestrutura é capaz de gerenciar, situação essa chamada de *overbooking* que pode resultar em maiores impactos no desempenho do sistema [Baset, S. A., et al, 2012].

O impacto de sistemas *multi-tenant* não é evidente a um usuário da nuvem, uma vez que esse possui acesso apenas ao estado virtual (e.g. informações do sistema operacional) da sua máquina virtual. Assim, como o usuário da nuvem geralmente não possui acesso direto ao hipervisor, ele não é capaz de consultar o estado físico dos recursos. Dessa maneira, uma máquina virtual pode apresentar um estado virtual disponível, porém pode não ser capaz de processar uma determinada demanda, caso possua seu estado físico degradado. Avaliar o desafio *multi-tenant* no momento da distribuição de demandas computacionais (e.g. *frameworks* de Big Data) pode melhorar o desempenho de aplicações que operam em nuvem computacional.

Adicionalmente, o provisionamento de recursos em tal contexto também deve considerar o estado físico do *cluster*. Uma vez que, caso o *cluster* encontre-se sobrecarregado computacionalmente e não haja mais recursos físicos disponíveis para virtualização, é necessário instanciar um novo conjunto de *hardware* para suprir essa demanda. Para tanto, torna-se necessário o desenvolvimento de uma solução que identifique a sobrecarga, tanto física quanto virtual, e acione o provedor de nuvem computacional visando a realização das operações de instanciação/desativação de um novo *cluster*.

Por outro lado, a partir do instante que diversos *clusters* estejam em operação torna-se necessário realizar o balanceamento de carga entre os mesmos, pois a aplicação tipicamente não realiza esse balanceamento.

1.2. Motivação e Hipótese

A variação de desempenho decorrente do compartilhamento de infraestruturas de computação em nuvem torna-se um fator crítico durante o processamento requerido por *frameworks* de *data streaming* para *Big Data*, devido ao seu processamento em tempo real. Nesse caso, se a capacidade de desempenho diminui, aplicações baseadas em *data streaming* podem não operar em velocidade suficiente para suprir suas demandas de tempo real. Reduzir

a variação de desempenho em tais infraestruturas de nuvem não é uma tarefa trivial, pois, tipicamente, não existe o controle sobre os recursos físicos por parte dos inquilinos que só podem gerir os recursos virtualizados para sua máquina virtual.

Um conjunto amplo de iniciativas se concentram em otimizar o gerenciamento dos recursos físicos e assim aprimorar a utilização de recursos virtualizados [Shen, Z., et al, 2011] [He, S., et al, 2012] [Tomas, L. e Tordsson, J., 2014], ou avaliar e minimizar a variabilidade de processamento, definindo ou instanciando um conjunto homogêneo de máquinas virtuais [Schad, J. et al, 2010] [Rego, P. A. L., et, al., 2011] [Galante, G., et al, 2012]. Outros trabalhos objetivam melhorar o desempenho das aplicações utilizando algoritmos de escalonamento que consideram a disponibilidade de recursos virtuais [Xu, J., et al, 2014] [Aniello, L., et al, 2013] [Grandl, R., et al, 2014] [Peng, B., et al, 2015]. Entretanto, os recursos físicos disponíveis e a alocação das máquinas virtuais estão sob controle dos provedores da infraestrutura de nuvem. Assim, a variação de desempenho causada pela utilização simultânea de diversos locatários sobre o mesmo *hardware* continua a ser um desafio a ser tratado na literatura.

Além disso, as cargas processadas por *frameworks* de *Big Data* podem sofrer alterações no decorrer do tempo, e esse comportamento dinâmico pode demandar uma quantidade maior de recursos computacionais em determinados períodos. Uma abordagem habitual na literatura para tratar esses casos é o incremento do número de nós (computadores) para o processamento no *cluster* [Segalin, D. et al, 2015]. Porém, o incremento no número de nós de um *cluster* aumenta a complexidade da infraestrutura de processamento e não resolve necessariamente a variabilidade de desempenho que as aplicações sofrem, uma vez que o nó adicional pode ser alocado em um *hardware* sobrecarregado. Adicionalmente, o aumento na quantidade de nós em um *cluster* requer que o ambiente de provisionamento de recursos da nuvem seja capaz de comunicar-se com o *framework* de *Big Data* para adicionar efetivamente um nó de forma virtual (a aplicação), tornando assim a solução específica à aplicação.

Uma abordagem que reduz a complexidade durante o provisionamento de recursos é denominada de provisionamento horizontal de recursos [Vecchiola, C., et al, 2012]. Essa abordagem duplica o *cluster* em vez de adicionar novos nós. Dessa forma, o ambiente de provisionamento de recursos da nuvem não necessita conhecer a aplicação [Verma, A., et al, 2011]. Embora o provisionamento horizontal reduza a complexidade no provisionamento de recursos, essa abordagem enfrenta outros desafios. A aplicação do cliente necessita conhecer

os diferentes *clusters* (com diferentes endereços) aptos a processar a carga e assim encaminhar a sua demanda computacional ao *cluster* mais adequado. No entanto, o processo de definição de qual *cluster* deve processar a demanda atual, a fim de realizar o balanceamento de carga (*Load Balance*), demanda um custo computacional elevado [Bohn, C. A. e Lamont, G. B., 2002]. De maneira geral isso não é tratado de forma transparente, necessitando de alterações na aplicação do cliente ou na infraestrutura da nuvem computacional [Fang, Y., et al, 2010].

Uma abordagem que permite a realização de forma transparente do balanceamento de carga é através da aplicação do modelo de Redes Definidas por Software (*Software Defined Network* - SDN) [Handigol, N., et al, 2009] [Handigol, N., et al, 2010] [Wang, R., et al, 2011]. O modelo SDN centraliza o gerenciamento da rede em uma entidade nomeada *Controller*, que define a topologia de rede em nível de *software* [Kreutz, D., et al, 2015]. A vantagem do modelo SDN, em relação ao modelo tradicional de redes, é a sua capacidade de definir em tempo real a topologia da rede, eliminando a utilização de *hardwares* de encaminhamento legados que utilizam políticas de encaminhamento estáticas [Kreutz, D., et al, 2015]. De maneira geral, quando a literatura trata o balanceamento de carga com a utilização do modelo SDN, os autores calculam o estado do *link* (rotas congestionadas) e recursos virtuais do nó (tipicamente CPU e memória) [Handigol, N., et al, 2009] [Handigol, N., et al, 2010].

As abordagens citadas não tratam do desafio *multi-tenant* dessa forma e as soluções acabam não sendo capazes de identificar se uma máquina virtual está realmente apta a processar uma demanda computacional baseada no recurso físico disponível. Logo, em determinada situação onde um *cluster* encontra-se com recursos físicos esgotados, a carga de processamento poderia ser encaminhada sem que o *cluster* seja capaz de processar tal demanda.

A hipótese deste trabalho é a viabilidade de tratar o impacto do *multi-tenant* de forma independente de hipervisor, desencadeando o aumento de desempenho e minimizando a variabilidade de processamento para aplicações *Big Data* de tempo real. De forma adicional, considera-se utilizar uma tecnologia emergente dentro da área de gerenciamento de redes chamada *Network Function Virtualization* (NFV), alinhada ao modelo SDN para provisionar recursos computacionais sob demanda e realizar o balanceamento de carga de forma transparente, de acordo com o estado físico e virtual das máquinas virtuais responsáveis pelo processamento das demandas de *data stream*.

1.3. Objetivo Geral

O objetivo geral deste trabalho é prover uma política de escalonamento de demandas computacionais para *frameworks* de tempo real (*data stream*) para *Big Data* que atuam em infraestruturas de nuvem computacional no modelo *multi-tenant.* Adicionalmente, prover uma técnica para provisionamento de recursos computacionais através de solicitações ao provedor do serviço de nuvem, para o provisionamento de recursos de forma horizontal. Finalmente, prover um *Load Balancer* que opere na distribuição das demandas computacionais de *data stream* entre diversas infraestruturas de nuvem computacional.

1.3.1. Objetivos Específicos

De forma a cumprir com o objetivo geral do trabalho, essa subseção apresenta os seguintes objetivos específicos:

a) Definir uma política de escalonamento que avalie o poder computacional físico e virtual dos nós do *cluster* em nuvem.

b) Definir uma política de reescalonamento (para nós já em execução, mas que provieram a entrar em estado *multi-tenant*) que avalie o poder computacional físico e virtual dos nós e efetue a migração da demanda em operação para nós com maior poder computacional disponível no *cluster*.

c) Definir uma solução de provisionamento de recurso de forma transparente à aplicação, que se comunica com o provedor de serviços da nuvem computacional para solicitar a instanciação e/ou o término de recursos computacionais (*cluster*).

d) Definir uma solução de *Load Balancer* que tem como objetivo distribuir a carga computacional entre diversos *clusters* de computação em nuvem de forma transparente ao usuário e à aplicação.

e) Definir um cenário e avaliar a proposta em relação às abordagens tradicionais encontradas na literatura.

1.4. Contribuição

As principais contribuições deste trabalho são listadas a seguir:

- Um mecanismo de escalonamento e reescalonamento dinâmico de demandas computacionais de *frameworks* de *Big Data* de tempo real. Essa solução tem como principal atribuição a seleção de nós menos sobrecarregados fisicamente e virtualmente, esses pertencentes ao *pool* de nós disponíveis num mesmo *cluster* de nuvem computacional.

- Uma abordagem para provisionamento elástico e transparente de recursos através da implementação de uma solução baseada em NFV incorporada à rede SDN. Essa solução é capaz de fornecer recursos de forma horizontal a fim de aumentar o poder computacional disponível para o *framework* de *data stream* para Big Data.

- Uma solução de *Load Balance* transparente, que atua como um módulo NFV que utiliza os dados provenientes do controlador SDN para o balanceamento. Essa solução é capaz de balancear a carga entre múltiplos *clusters* de acordo com seu estado computacional atual. Para computar o estado computacional do *cluster* é utilizado o estado físico e virtual de cada nó (máquina virtual) do *cluster* de nuvem computacional.

- Uma solução adicional baseada em aprendizagem de máquina capaz de identificar impacto *multi-tenant* na aplicação, através da análise da utilização dos recursos virtuais e do desempenho da aplicação monitorada. A abordagem é independente de hipervisor e do provedor do serviço de nuvem computacional, atuando no domínio do cliente sem acesso às métricas de recursos físicos.

1.5. Organização

O restante deste documento encontra-se organizado da seguinte forma: O Capítulo 2 trata da fundamentação teórica. O Capítulo 3 discute os trabalhos relacionados. O Capítulo 4 apresenta a proposta do trabalho. O Capítulo 5 apresenta o protótipo e as avaliações. Para concluir o documento o Capítulo 6 apresenta as considerações finais.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica relacionada ao processamento Big Data, Nuvem Computacional (*Cloud Computing* - CC) e Redes Definidas por Software (*Software Defined Network* - SDN).

2.1. Big Data

Incontáveis fontes são responsáveis por produzir uma grande massa de dados, seja através de usuários interconectados via internet, corporações que necessitam armazenar em larga escala os dados referentes a seus clientes e a seus fornecedores, ou ainda através de redes sociais com milhões de acessos e novos conteúdos adicionados a cada segundo. Essa produção de massa de dados ocorre em todos os setores da economia, tornando necessárias abordagens computacionais para tratar tal contexto que tipicamente é tratado pela literatura como o "fenômeno *Big Data"* [Chen, M. et al, 2014].

Um estudo reportado pela *International Data Corporation* (IDC) em 2011 apontou que a quantidade total de dados produzidos e copiados no mundo era de 1.8 zettabytes [Gantz e Reinsel, 2011]. Esse número aumentou cerca de nove vezes nos cinco anos seguintes ao estudo, e a tendência é dobrar pelo menos a cada dois anos [Chen, M. et al, 2014].

Segundo [Chen, M. et al, 2014], as características da Big Data podem ser sumarizadas através de quatro V's: (i) Volume: devido à crescente quantidade de dados; (ii) Variedade: faz relação aos diversos tipos de dados, que incluem dados semiestruturados e não estruturados como texto, vídeo, áudio e web, além de dados estruturados tradicionais. (iii) Velocidade: relacionado à atualização dos dados em tempo oportuno, de modo a utilizar o máximo do valor

comercial que as tecnologias de Big Data proveem e (iv) Valor: onde o custo envolvido na coleta, armazenamento e processamento da análise seja compensado ao fim do processo.

O modelo Big Data pode ser dividido em cinco categorias para melhor compreensão de suas características (Figura 1): (i) fonte de dados, (ii) formato de conteúdo, (iii) armazenamento de dados, (iv) preparação dos dados e (v) processamento de dados [Hashem, I. A. T., et al, 2015].

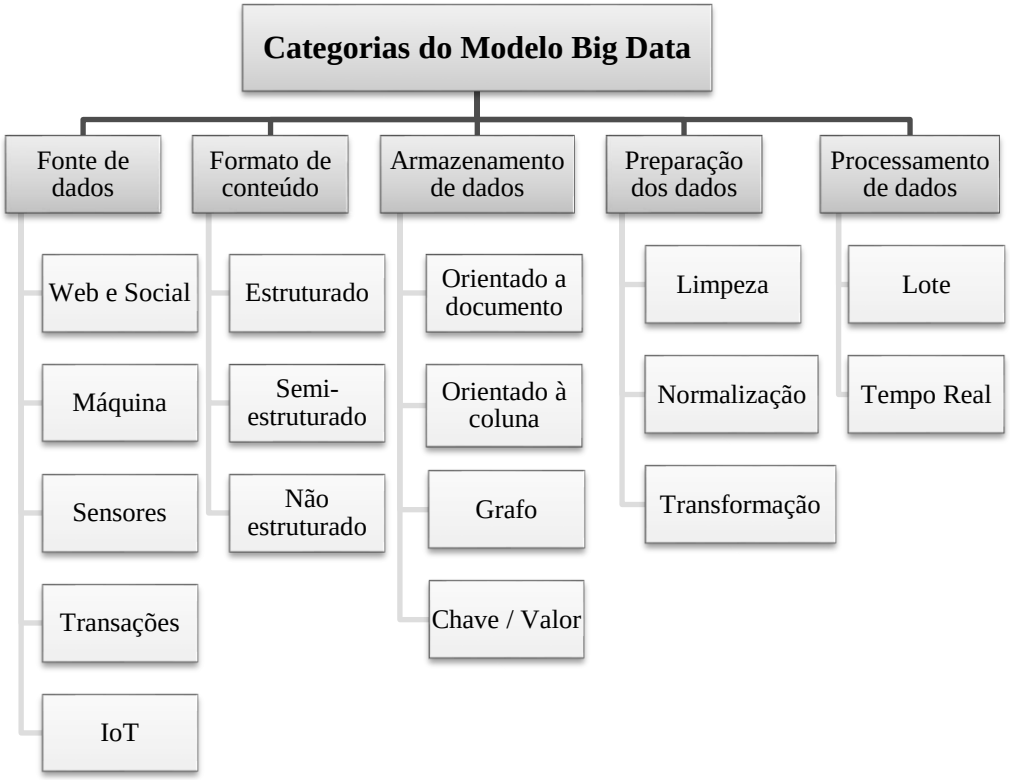


Figura 1 – Classificação Big Data. Adaptado de [Hashem, I. A. T., et al, 2015].

O tratamento dessa grande massa de dados implica no desenvolvimento de infraestruturas de sistemas distribuídos. Nesse sentido, um conjunto de técnicas e tecnologias surgem para o tratamento adequado dos desafios da Big Data, conforme relatado a seguir.

2.1.1. Framework para processamento de Big Data modo batch

No contexto de *frameworks*, que operam em modo *batch* e satisfazem as características que sistemas *Big Data* necessitam, encontram-se Hadoop [Apache, 2016], MapReduce [Dean and Ghemawat, 2008], Dryadv [Isard M. et al. 2007], Pregel [Malewicz G, et al. 2010], entre outros. Entretanto, a literatura trata de forma exaustiva o *framework* Hadoop.

Hadoop se destaca como um das principais tecnologias envolvidas no contexto de processamento para Big Data. *Hadoop* é um *software* de código aberto, que realiza computação distribuída de grandes conjuntos de dados de modo confiável e escalável, e foi desenvolvido para operar em um aglomerado de computadores (*cluster*).

a) *Hadoop* - Visão Geral

O *framework* Hadoop emprega o modelo de programação MapReduce [Dean, J. and Ghemawat, S., 2008] que se baseia em duas primitivas: *Map* e *Reduce* (Mapear e Reduzir), através do conceito de chave/valor. Logo, a função *Map* recebe um conjunto de chave-valor $\{K_1, V_1\}$ e gera como saída também uma lista de chave/valor $\{K_2, V_2\}$. Esses pares de chave/valor são definidos com base na implementação da função Map, e ao término de cada Map um nó central coleta a saída e ordena os valores pela chave, e em seguida as chaves ordenadas são distribuídas para as tarefas *Reduce*. Pares de chave/valor que possuem a mesma chave são executados para a mesma tarefa *Reduce*. Essa recebe todos os valores de V_2 e uma mesma chave K_2 para processar uma saída de chave/valor $\{K_3, V_3\}$, formando o procedimento *MapReduce*. O modelo *MapReduce* utiliza uma arquitetura *Master/Slave*, onde o nó *Master* detém diversas atribuições como: criar o número de tarefas e escalonar as tarefas para os nós *Slaves*, e esses realizam o processamento da demanda. A arquitetura *Master/Slave* não é característica exclusiva do Hadoop e as demais tecnologias de Big Data também seguem esse modelo. A Figura 2 ilustra o modelo de programação MapReduce.

O Hadoop tem foco em aproveitar a estrutura e os recursos que um aglomerado de computadores proveem, de modo que suas aplicações utilizem ao máximo os recursos disponíveis. O Hadoop tem como característica dois principais mecanismos: (i) Sistema de Arquivos Distribuído (*Hadoop Distributed File System* – HDFS), que tem como objetivo realizar a fragmentação, espalhamento e replicação dos dados necessários dentre os nós formadores do aglomerado, e (ii) o modelo de programação MapReduce, que permite a divisão

de uma aplicação em diversas tarefas com tamanhos menores, de forma que cada tarefa seja executada da maneira mais distribuída possível. Ambos os mecanismos – HDFS e MapReduce – foram desenvolvidos para permitir o gerenciamento de falhas de forma automática pelo *framework* [Apache, 2016].

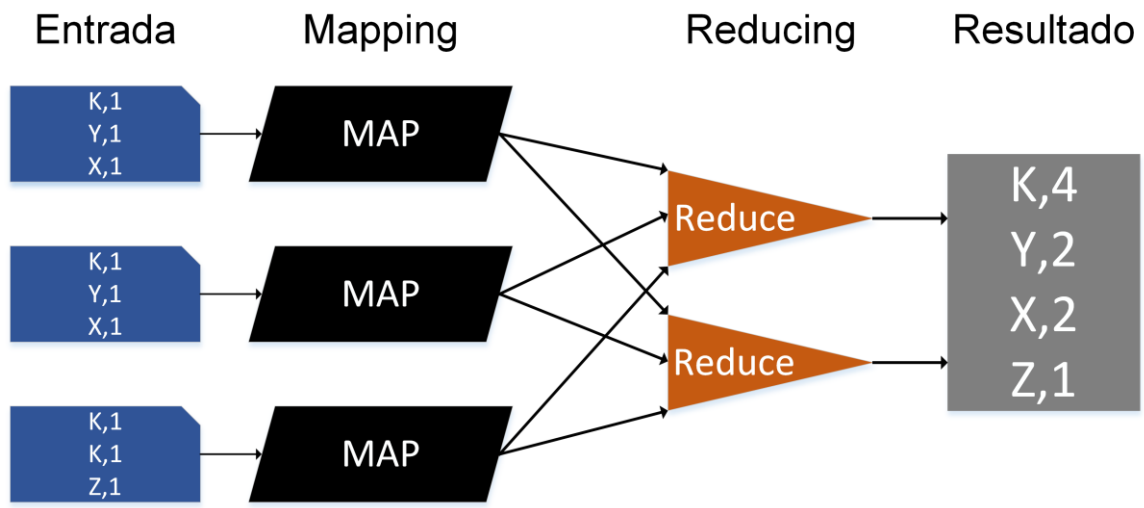


Figura 2- Modelo de programação *MapReduce*

É interessante ressaltar que Hadoop e MapReduce operam com processamento modo *batch*, ou seja, implica que todo o dado necessário para processamento seja previamente armazenado no sistema de arquivos do Hadoop, no caso, o HDFS.

2.1.2. *Framework* para processamento de Big Data modo *stream*

O modo *batch* é uma abordagem funcional para processamento de *Big Data*, porém não é adequado para situações onde o tratamento de uma grande massa de dados deve ocorrer em tempo real. Inúmeras aplicações produzem dados em tempo real, e esses dados são encaminhados para servidores que realizam o devido processamento referente a cada aplicação.

Um exemplo clássico e conhecido na literatura de emprego do processamento em tempo real foi o recorde de *tweets* registrado em 2013 durante a exibição filme de animação "*Castle in the Sky*" no fim de semana de 02 de agosto no Japão, onde a marca alcançou 143.199 *tweets* por segundo [Twitter record, 2013], tipicamente em dias normais são emitidos mais de 500 milhões de *tweets*, representando 5.700 *tweets* por segundo.

Outro exemplo de aplicação que necessita de processamento em tempo real são as redes de sensores onde, por exemplo, os dados referentes ao clima de uma determinada região são lidos e repassados para processamento, ou ainda aplicativos que utilizam sensores GPS via telefonia móvel, para informar a localização de consumidores e as condições de tráfego do trânsito local.

O trabalho de [Stonebraker et al, 2005] apontou os principais requisitos que os sistemas de processamento *stream* devem apresentar: (i) Mobilidade de dados: para manter uma baixa latência, é fundamental que o sistema efetue o processamento sem a necessidade de uma operação custosa como o armazenamento dos dados que serão processados, logo espera-se que um sistema *stream* proporcione que os dados se mantenham em movimento. (ii) Consulta de dados: são necessários mecanismos de consulta de dados a fim de obter valores já processados, bem como calcular análises em tempo real. (iii) Tratamento de imperfeição de *stream*: a arquitetura deve tratar as diferentes quantidades de dados que podem variar a todo instante, e também a chegada dos dados fora de ordem. (iv) Garantia de resultados previsíveis: o mecanismo para processamento de *stream* deve garantir resultados previsíveis e repetíveis. (v) Integração de armazenamento e *stream*: diversas aplicações necessitam de comparações de dados do "presente" com dados do "passado" e para isso é esperado que o armazenamento desses ocorra de forma eficiente para sua posterior utilização. (vi) Garantia de segurança dos dados e disponibilidade: Garantir alta disponibilidade da aplicação e a integridade dos dados durante todo processamento, mesmo com ocorrência de falha durante processo. (vii) Escalabilidade: distribuir o processamento por todo o aglomerado de computadores buscando escalabilidade de forma incremental. (viii) Processamento e resposta instantânea: combinado com todos os requisitos listados anteriormente, o sistema deve garantir resposta em tempo real para aplicações de alto volume de dados.

Uma abstração conceitual para *stream* relatada em [Zhang, Z. et al. 2010] é que todo o processamento ocorre em ambiente compartilhado, onde diversas fontes podem enviar demandas computacionais. Essa demanda é recebida por determinadas entidades chamadas de Elemento de Processamento (EP) / (*Processing Element - PE*). Os EPs podem ser alocados em diferentes máquinas, formando um conjunto de EPs que são responsáveis por consumir e tratar todo o trabalho demandado. A distribuição dos EPs ocorre via um componente de escalonamento que baseado nas políticas típicas de cada ferramenta realiza o escalonamento. A Figura 3 representa essa abstração.

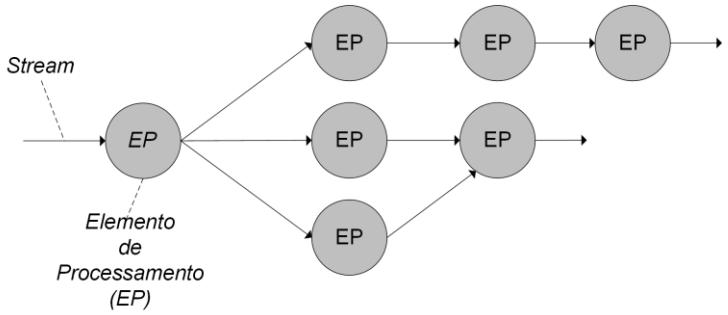


Figura 3 – Representação conceitual de processamento *stream*, baseada em [Zhang, Z. et al. 2010].

Dentre as ferramentas para processamento de *stream* referenciadas na literatura, encontram-se: Apache Storm [Toshniwal, A. et al., 2014], [Storm, 2016], S4 [S4, 2013], MillWheel [Akidau, A. et al. 2013], Samza [Feng, T., et al. 2015], Spark Streaming [Han, Z. and Zhang, Y., 2015] e Photon [Ananthanarayanan, R. et al. 2013]. O Apache Storm é uma ferramenta para processamento de *stream*, amplamente referenciado na literatura e apresenta as características que o modelo de tempo real deve conter. Além disso, é utilizado em produção por grandes corporações como: Yahoo Japan [Yahoo Japan, 2016], Yahoo [Yahoo, 2016], Twitter [Twitter, 2016], Spotify [Spotify, 2016], dentre outras. A seguir são descritas as características do *framework* Apache Storm.

a) *Apache Storm* - Visão Geral

No quesito processamento de *stream* uma tecnologia que se destaca na literatura é o *Apache Storm* [Storm, 2016], que é uma ferramenta para computação distribuída de tempo real e de código aberto. *Storm* é rápido, escalável, tolerante a falhas e garante o processamento dos dados. Essa garantia é alcançada ao anexar um identificador de 64bits, gerado de maneira aleatória, para cada mensagem recebida. Ao final do processamento ocorre uma validação via mecanismo reverso da ferramenta que verifica se todos os identificadores foram processados [Toshniwal, A. et al. 2014]. O processamento de dados no *Storm* ocorre através da definição de topologias que consomem o *stream* de dados e realizam o processamento de forma arbitrária. Topologias definem o formato da infraestrutura necessária para realizar o processamento sobre as fontes de dados. O processamento de uma topologia é realizado por dois componentes principais os *Spouts* e *Bolts*. Os *Spouts* são responsáveis pela leitura de dados de fontes (internas ou externas) gerando *streams* de dados para a plataforma, enquanto os *Bolts* são responsáveis por consumir os dados gerados pelos *Spouts* e efetuar o devido processamento. Cada unidade

de processamento gerada pelos *Spouts* é denominada tupla (mensagem), uma sequência de tuplas da topologia caracteriza o *stream* de dados.

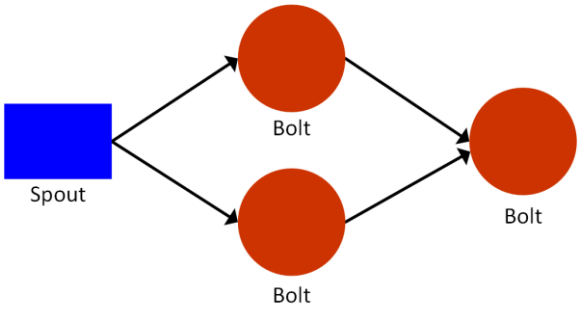


Figura 4 - Topologia Storm - Visão Lógica

Uma topologia é composta por diversos executores (*threads*) que realizam o processamento predefinido de um *Spout* ou *Bolt*. A topologia pode ser compreendida como um grafo direcionado acíclico, determinando o fluxo de dados e de processamento. A topologia em produção, formada pelos *Spouts* e *Bolts*, é entendida na literatura como abstração lógica do ambiente *Storm* (Figura 4).

A arquitetura do *Storm*, chamada de *Storm Cluster* (Figura 5), é composta por um servidor chamado *master node* e de um ou mais *worker nodes*[1]. O *master node* executa o processo (*daemon*) *Nimbus*, responsável por submeter topologias que serão executadas nos *worker nodes*. Cada *worker node* possui um processo (*daemon)* denominado *Supervisor*, responsável por gerir as tarefas (baseadas nas topologias) submetidas pelo *Nimbus*. Além do *Supervisor*, o *worker node* possui *slots* que são portas de comunicação disponíveis. Normalmente, a quantidade de *slots* disponíveis para cada *worker node* é associada ao número de núcleos da CPU. Para processar as tarefas que chegam no *slot*, é necessário associar a um *worker* (processo java) que possui um ou mais executores (*threads*).

Os *Supervisors* realizam comunicações periódicas (*heartbeat*) com o *Nimbus*, informando as topologias em execução e os *slots* disponíveis para execução de novas demandas.

[1] Atenção para distinção entre **(i) *worker*** *e* ***(ii) worker node***. **(i) *worker*** é um processo Java, composto pelos executores de uma topologia. **(ii) *worker node*** é uma máquina física que faz parte do Storm Cluster.

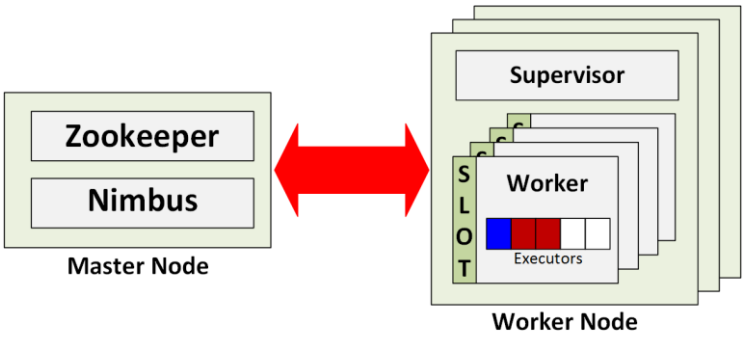


Figura 5 – Storm Cluster

A comunicação entre o *master node* e os *worker nodes* acontece através do *Apache Zookeeper* [Zookeeper, 2016]. *Zookeeper* é uma ferramenta pertencente à fundação *Apache* que provê um serviço de configuração, coordenação e descoberta de serviços em aplicações distribuídas. Esse esquema é chave para resiliência da arquitetura *Storm.* Caso o *Nimbus* venha a falhar, os *workers* continuam com a evolução do trabalho, e de forma adicional os *Supervisors* podem reiniciar os *workers* que venham a falhar.

b) Escalonamento de tarefas Storm

O processo de escalonamento do *Storm* define como os executores serão dispostos na infraestrutura disponível. Toda nova topologia submetida necessita ser escalonada para que os *worker nodes* executem seu processamento. Para isso, o *Nimbus* utiliza por padrão o escalonador de topologias *EvenScheduler*. O *EvenScheduler* utiliza como estratégia de escalonamento a política do algoritmo *round-robin*, e essa política distribui de forma uniforme os *executors* aos *workers* alocados para a topologia.

Para ilustrar a execução dessa política, será utilizado como exemplo uma topologia padrão chamada *Word Count Topology* [Word Count Topology, 2013] que objetiva contar palavras em frases emitidas interruptamente, composta por:

- *Spout* formado por 5 *executors*, operando como fonte de dados da topologia, através da geração periódica de frases.
- *Bolt* formado por 8 *executors*, denominado *Split*. Responsável pela leitura das frases geradas pelo *Spout*, e pela separação em palavras para envio ao próximo componente.

- *Bolt* formado por 12 *executors*, chamado de *count*. Responsável por realizar a contagem das palavras originadas do *Split*.

A Figura 6, representa um *Storm Cluster* que contém: 1 *master node* e 2 *workers nodes*, sendo que cada *worker node* possui 2 *slots*, e cada *slot* aloca 1 *worker*. A topologia do *Word Count* é formada por 25 *executors* que serão distribuídos em 4 *workers* através do *EvenScheduler*. O *Nimbus* possui uma fila de topologias que aguardam para ser executadas baseadas nas políticas do escalonador. Neste caso, a primeira topologia a ser submetida é a *Word Count Topology* (evento 1 e 2).

Ao analisar a Figura 6, é possível identificar o comportamento da política aplicada pelo *EvenScheduler* no momento da distribuição dos *executors* aos *worker nodes*. Os 25 *executors* são distribuídos de forma que os *Slots* disponíveis para a topologia sejam ocupados, independentemente de seu estado computacional. Em resumo, o *EvenScheduler* seleciona o primeiro *executor* e o aloca no primeiro *slot* disponível. Na sequência, seleciona o segundo *executor* e aloca no próximo *slot*, e repete esse procedimento para todos os *executors* restantes da topologia. Embora a estratégia utilizada no *round-robin* seja interessante por sempre distribuir uniformemente os *executors* para os *worker nodes*, nem sempre essa abordagem é eficiente. Em ambientes *multi-tenant*, aplicar a política de escalonamento igualitária sem considerar o estado físico da máquina pode acarretar em impactos negativos de desempenho de execução das topologias.

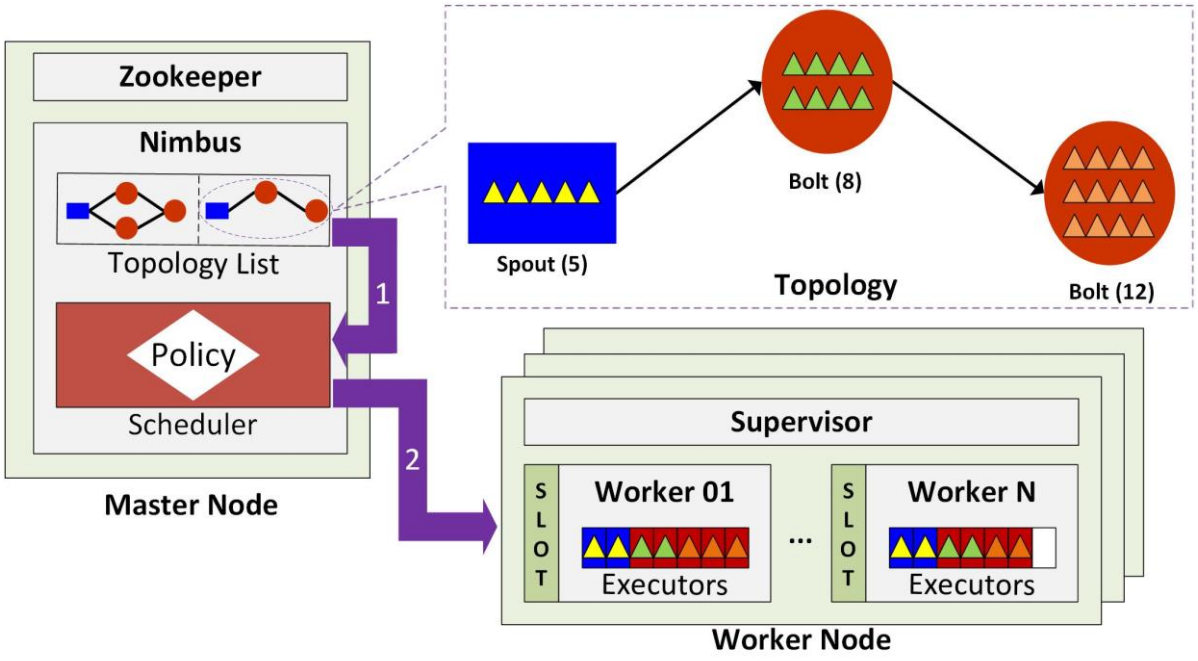


Figura 6 - Visão geral da arquitetura do *Storm* e o escalonamento da topologia WC

O *Storm* permite a customização das políticas de escalonamento, conforme as necessidades do usuário/ambiente. A interface que provê esta customização é chamada de *IScheduler* e possui um método de escalonamento que necessita de dois parâmetros: (i) um objeto que contenha a definição de todas as topologias em execução; (ii) um objeto que representa o *cluster*, com todas as informações sobre *worker* nodes, *slots* e alocações atuais. Logo, através dessa interface, torna-se possível implementar um escalonador customizado.

Um fator significativo a se destacar no contexto de *frameworks* para o processamento *Big Data* é a infraestrutura computacional necessária para tratar de forma distribuída o alto volume de dados heterogêneos. Independente da abordagem utilizada pelo *framework* (*batch* ou *stream*), essa infraestrutura de *hardware* deve ser adequada para receber a demanda computacional gerada através de ferramentas de processamento *Big Data*. Nesse sentido, ambientes de Computação em Nuvem (*Cloud Computing*) têm sido amplamente empregados no processamento das aplicações de *Big Data*, pois é uma alternativa que fornece requisitos necessários para uma grande massa de dados como: grande capacidade de armazenamento, escalabilidade, elasticidade, desempenho elevado e alta disponibilidade [Agrawal et al. 2011].

2.2. Computação em Nuvem

Uma característica indispensável no contexto de ferramentas que tratam de forma direta uma quantidade elevada de dados é a inevitabilidade de uma infraestrutura de *hardware* que suporte tal demanda. Nesse sentido, a Computação em Nuvem (*Cloud Computing*) é amplamente adotada para o processamento *Big Data*. A ideia central da computação em nuvem é prover elasticidade de recursos computacionais através de uma infraestrutura dedicada para tanto. A seguir serão relacionados os conceitos e características que delineiam esse modelo computacional.

2.2.1. Definição NIST para Computação em Nuvem

O conceito de computação em nuvem mais difundido na literatura foi elaborado pelo *National Institute of Standards and Technology* (NIST) [Mell, Grace, 2011]. Esse delimita o modelo em três camadas: atributos de serviço, modelo de serviços e modelos de implantação. A Figura 7 ilustra o referido modelo.

O NIST elenca cinco características essenciais que o modelo de computação em nuvem deve possuir, são eles: (i) autoatendimento, (ii) amplo acesso à rede, (iii) conjunto de recursos (*Pool*), (iv) elasticidade e (v) medição de recursos (contabilização).

A definição do NIST elenca três modelos de serviço (Figura 7): (i) Software como Serviço (*Software as a Service* – SaaS), que provê serviços computacionais para o consumidor final, geralmente caracterizado por uma aplicação já em operação pronta para utilização. (ii) Plataforma como Serviços (*Platform as a Service* – PaaS), modelo que fornece recursos para que o consumidor implante suas aplicações em software, com a necessidade de utilizar um conjunto de ferramentas e linguagens de programação suportadas pelo provedor. (iii) Infraestrutura como Serviço (*Infrastructure as a Service* – IaaS), fornece um sistema computacional básico formado por processadores, memória, armazenamento e rede, e possibilita que o consumidor implante uma variedade de programas – Sistemas Operacionais, *frameworks* de desenvolvimento e aplicativos em geral – formando assim seu ambiente de trabalho de forma autônoma.

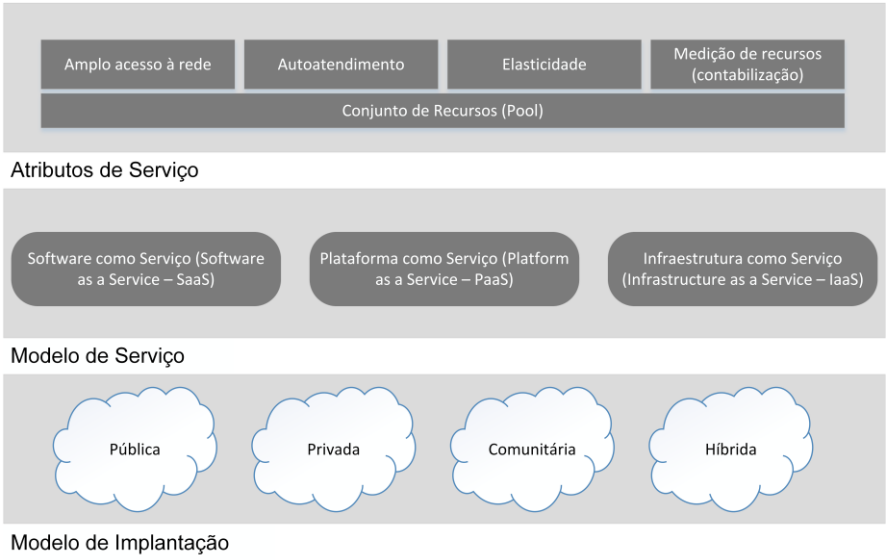


Figura 7 – Modelo NIST para computação em nuvem. Adaptada de [Mell, Grace, 2011].

O NIST ainda define quatro modelos para implantação de nuvens computacionais (Figura 7), conforme segue: (i) Pública: fornece acesso aos recursos computacionais conforme a demanda dos consumidores. É interessante destacar que o termo "público", não significa acesso gratuito aos recursos, mas sim que qualquer público pode ter acesso, desde que cumpra as regras de utilização definidas pelo fornecedor do serviço. (ii) Privada**:** infraestrutura é operacionalizada de forma exclusiva para uma organização. Sua administração pode ser realizada pela própria organização ou por uma entidade terceira. (iii) Comunitária**:** permite que diferentes organizações com interesses comuns utilizem o recurso de forma compartilhada. (iv) Híbrida**:** é a composição de dois ou mais modelos de nuvem que, de forma geral, são múltiplos modelos de nuvem conectadas entre si.

2.2.2. Virtualização e Multilocatário (Multi-Tenant)

Virtualização é um mecanismo presente e largamente utilizado no modelo de computação em nuvem [Grobauer et al. 2010]. A virtualização permite a flexibilização dos recursos de *hardware* e a criação de ambientes de processamento independentes e isolados, capazes de ser instanciados e encerrados sob demanda. Essa flexibilização proporciona que recursos como CPU, memória e disco da máquina física façam-se compartilhados entre diversas

máquinas virtuais (*Virtual Machine* - VM), que executam suas demandas de forma independente e isolada de outras máquinas virtuais. Essa alocação de recursos ocorre através do monitor de máquinas virtuais (*Virtual Machine Monitor* – VMM) responsável por intermediar a comunicação da máquina virtual com os recursos físicos disponíveis [Smith, J. e Nair, R., 2005].

A virtualização de produtos e serviços computacionais tornou-se viável em decorrência da aplicação do conceito de multilocatário (*multi-tenant),* que possibilita aspectos como o compartilhamento de recursos de *hardware*. Dessa forma, o *multi-tenant* está intrinsecamente ligado ao modelo de computação em nuvem. A camada de virtualização, também denominada de Hipervisor, é responsável pela construção das interfaces virtuais a partir da infraestrutura real [M. Rosenblum e T. Garfinkel, 2005].

A Figura 8 ilustra o conceito da virtualização do *hardware* por um Hipervisor. Os recursos físicos da máquina (e.g. processamento, disco, rede, entre outros) são abstraídos e acessados através do Hipervisor, e qualquer máquina virtual que deseja acessar um recurso físico requisita ao Hipervisor o acesso ao *hardware*.

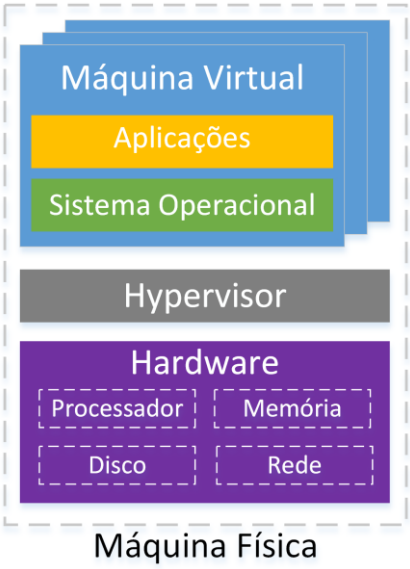


Figura 8 - Ambiente Compartilhado

O acesso da máquina virtual ao recurso físico do seu hospedeiro é específico ao Hipervisor utilizado. Por exemplo, o *Hypervisor* VMWare [VMware, 2016] por padrão compartilha a CPU física entre as máquinas virtuais através do algoritmo de *fair share*, que

fornece a CPU para as máquinas virtuais de acordo com seu histórico de utilização. Por outro lado, o *Hypervisor* Xen [Xen Project, 2016] fornece tempos iguais de acesso à CPU independentemente do histórico de utilização[2].

A capacidade de processamento de uma máquina virtual é estritamente relacionada à maneira como os recursos físicos estão sendo utilizados [Smith e Nair, 2005]. Recursos como o acesso à rede, à leitura e à escrita em disco são afetados pelo acesso concorrente, fator ocorrente em ambientes *multi-tenant.*

Utilizar a virtualização permite aos provedores de computação em nuvem compartilhar o uso de um mesmo *hardware* entre várias máquinas virtuais. Isso ocasiona na situação onde locatários (*tenants*) distintos podem concorrer pelos mesmos recursos físicos do *hardware* de forma simultânea, o que pode implicar na queda de desempenho das tarefas entre todos os locatários.

Conforme já mencionado anteriormente, ambientes de nuvem computacional fornecem a infraestrutura necessária para que *frameworks* de processamento para *Big Data* executem suas demandas. Nesse contexto, o conceito de *multi-tenant*, que é característico das nuvens computacionais, torna-se um desafio a ser tratado por essas ferramentas. É pertinente que os *frameworks* de *Big Data* avaliem previamente o estado físico dos *hardwares* que hospedam suas aplicações no momento do escalonamento das demandas computacionais, de modo a diminuir a queda de desempenho de suas tarefas quando alocadas para um *hardware* já comprometido por outros locatários.

2.3. Redes Definidas por Software e Nuvens Computacionais

A infraestrutura que os provedores de computação em nuvem fornecem são de extrema importância para as demandas computacionais recorrentes no cenário *Big Data*. O modelo

[2] A forma de acesso ao recurso físico normalmente é definida de acordo com as necessidades do administrador do *cluster*, e pode sofrer variações de acordo com a versão do *Hypervisor* utilizado.

multi-tenant, típico das nuvens computacionais, permite a um consumidor (locatário) alugar um número de máquinas virtuais conforme a necessidade da aplicação.

Quando um locatário possui mais de uma máquina virtual em nuvem, é possível estabelecer uma rede local para o processamento das demandas. O número de máquinas virtuais dessa rede pode ser ampliado ou reduzido, conforme necessidade do locatário, devido ao conceito de elasticidade que é característico do modelo de nuvem.

De maneira geral, os provedores de nuvem pública não fornecem acesso aos comutadores de rede [Paniagua, D., 2016]. Nesse contexto é possível realizar uma sobreposição SDN, também mencionada na literatura como "*SDN overlay*" [Kreutz, D., et al, 2015], que se resume em um método de implementação que virtualiza uma rede SDN logicamente separada e sobreposta à infraestrutura física existente.

Para situações onde o poder de processamento não atende a demanda em tempo hábil, existe a necessidade de iniciar um novo *cluster* com novo conjunto de máquinas virtuais para suprir essa situação. Nesse sentido é interessante a utilização de tecnologias que detenham o controle do estado da rede para uma tomada de decisão assertiva, características essas encontradas no modelo Redes Definidas Por Software (*Software Defined Networks* – SDNs).

O SDN realiza o desacoplamento do plano de controle do plano de dados da rede [Kreutz, D., et al, 2015]. O plano de controle contém a lógica responsável pelo estabelecimento das regras de encaminhamento de pacotes, enquanto o plano de dados aplica em tempo real as regras previamente programadas no plano de controle que são aplicadas para todos os pacotes recebidos pelos comutadores da rede. Esse desacoplamento transforma os *switches* de rede em dispositivos de encaminhamento simples, enquanto a lógica de controle é implementada em um controlador que opera como um sistema operacional de rede centralizado. Os dispositivos de encaminhamento (plano de dados) possuem um conjunto de instruções (regras para fluxos) que são aplicadas sobre os pacotes da rede. De maneira objetiva, o plano de dados contém uma tabela de fluxo (*flow table*) e uma ação associada a cada entrada (*flow entry*), que pode ser gerenciada (alterada ou removida) pelo plano de controle (*controller*). Para gerenciar as entradas de fluxo, o controlador necessita de um protocolo para comunicação entre planos controle e plano de dados, que instale, modifique ou remova os fluxos no dispositivo de rede. Nesse sentido, uma solução emergente na literatura é o protocolo *OpenFlow*, proposto por [N. McKeown, et al. 2008] e padronizado pela *Open Network Foundation* [ONF, 2014], que

permite aos controladores o acesso e a modificação das regras de fluxo de forma remota nos dispositivos de rede (e.g.: *switches* e roteadores).

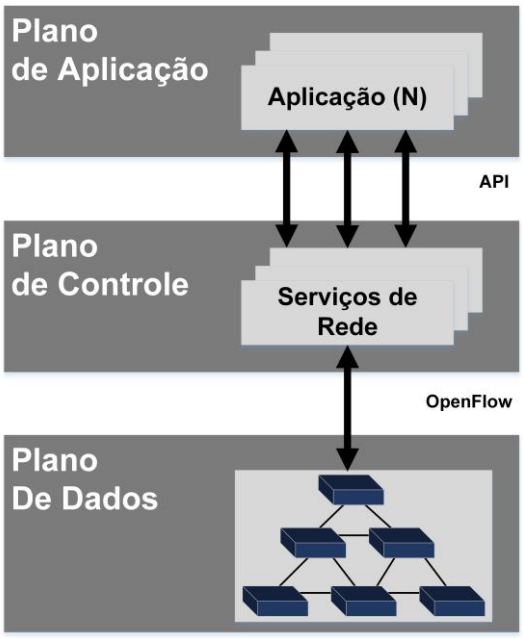


Figura 9 – Abstração do Modelo Redes Definidas por Software. Adaptado de ONF [ONF, 2014].

A Figura 9 representa uma abstração do modelo SDN através de 3 camadas: (i) Plano de Aplicação: em que rodam os sistemas e aplicativos dos consumidores. (ii) Plano de Controle: responsável pela lógica nas definições das regras de encaminhamento dos pacotes na rede. (iii) Plano de Dados: representado pelos dispositivos (*hardware*) de rede, responsáveis pelo encaminhamento de pacotes baseados nas regras definidas no plano de controle.

Características das SDNs – programabilidade da rede de forma ágil, centralizada e com a utilização de padrões abertos (*OpenFlow*) – despertam, na literatura, o interesse no desenvolvimento de soluções que concatenam o modelo de Computação em Nuvem com a arquitetura SDN [Wang, G. et al 2012], [Das A., et al, 2013], [Ferguson, A., 2013], com o objetivo de otimizar serviços e prover novas abordagens tecnológicas.

A seção seguinte apresenta uma visão geral sobre o protocolo *OpenFlow* e respectivos controladores, que são elementos intrínsecos ao SDN. Na sequência, é apresentado o conceito de NFV - *Network Function Virtualization*, que é uma tecnologia de virtualização de aplicações computacionais que pode ser combinada ao SDN promovendo maior produtividade.

2.3.1. OpenFlow

O protocolo *OpenFlow* (OF) é um protocolo de padronização entre a comunicação dos dispositivos de encaminhamento (*switches)* e o elemento controlador (*controller*). Na arquitetura SDN, o elemento controlador é análogo a um sistema operacional de nível de rede. A principal motivação dos autores no desenvolvimento do protocolo era utilizar a infraestrutura de *hardware* já existente e em produção para testar novas abordagens de roteamento. Isso devido ao fato de que os códigos fontes dos dispositivos de rede em operação não podem ser modificados, causando o efeito chamado de ossificação da infraestrutura de rede [N. McKeown, et al. 2008]. Os autores utilizaram as características comuns das tabelas de fluxo dos *switches ethernet* para desenvolver o protocolo padronizado para gerenciamento remoto dessas tabelas através de *software*. Assim, o protocolo *OpenFlow* fornece recursos para gerenciar um dispositivo de rede sem a necessidade do código fonte dos ativos de rede [Lara, A., et al. 2013].

A arquitetura *OpenFlow* é formada por três componentes centrais: (i) *Switch OpenFlow*, (ii) Canal seguro para comunicação (*Secure Socket Layer* - SSL) e (iii) Controlador. Tal arquitetura é representada na Figura 10.

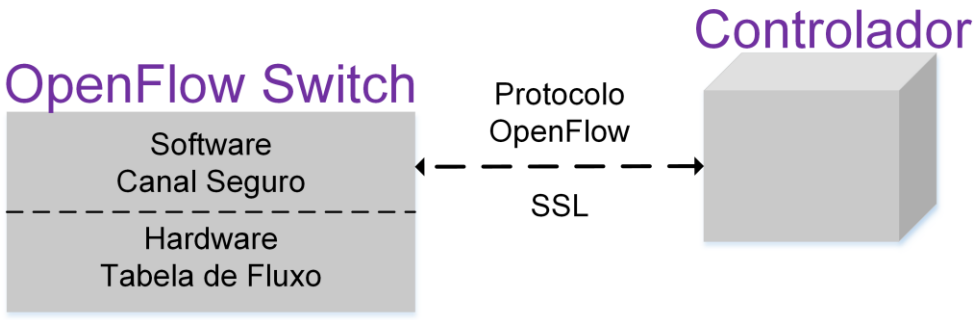


Figura 10 – Componentes da arquitetura OpenFlow. Adaptado de [N. McKeown, et al. 2008]

Os *switches* contam com tabelas de fluxo para tratar o encaminhamento dos pacotes. Essa tabela é uma lista de entradas de fluxos que contém campos de correspondência, contadores e instruções. Os pacotes são confrontados com os campos da tabela e processados de acordo com a ação contida na entrada, ou ainda o pacote pode ser encapsulado e enviado ao controlador.

O controlador é um *software* que faz uso do protocolo *OpenFlow* para manipular diretamente as tabelas de fluxos dos dispositivos da rede e toda sua comunicação ocorre via um canal seguro, que serve como interface de comunicação entre controladores e *switches* [Lara, A., et al. 2013]. Cada entrada na tabela de fluxo caracteriza um padrão que pode conter até dez campos de cabeçalhos que correspondem ao encontrados tipicamente em uma rede local, como: TCP, IP, Ethernet. A Tabela 1 ilustra os campos disponíveis na versão 1.0 do OpenFlow.

Porta de Entrada	ID da VLAN	Ethernet			IP			TCP	
		Origem	Destino	Tipo	Origem	Destino	Tipo	Origem	Destino

Tabela 1 - Campos disponíveis para tratamento de fluxos para o OpenFlow versão 1.0. Adaptado de [N. McKeown, et al. 2008]

Cada pacote recebido é confrontado com esses conjuntos de entrada, que formam um determinado padrão e, havendo completa equivalência entre a regra associada, executa-se uma ação. Dentre essas ações estão: (i) encaminhar os pacotes do fluxo em questão para uma porta (ou portas) específica; (ii) encapsular e encaminhar os pacotes do fluxo em questão para um controlador: isso normalmente ocorre no primeiro pacote de um novo fluxo e dessa forma é o controlador que avalia e associa esse pacote à tabela de fluxo; (iii) descartar os pacotes: essa ação pode ser aplicada no quesito segurança evitando ataques de negação de serviço ou redução do tráfego *broadcast* entre sistemas finais.

2.3.2. Controladores SDN

Controladores são entidades que implementam o plano de controle no SDN e podem aplicar, modificar ou remover as entradas na tabela de fluxo através do protocolo *OpenFlow*. Operam de forma análoga a um servidor externo que detém uma visão global e centralizada da rede, englobando *switches*, máquinas e todos os fluxos que trafegam nesse cenário. O controlador intitulado NOX [Gude, N., et al., 2008] é o pioneiro na literatura, foi desenvolvido pelos criadores do protocolo *OpenFlow* e opera como um *Hypervisor* de Rede.

Os principais componentes presentes em uma rede SDN baseada em NOX são: um conjunto de *switches*, um ou mais servidores conectados em rede e um banco de dados centralizado que contém a visão geral da rede, conforme modelo ilustrado pela Figura 11 . Esses

servidores executam o *software* NOX e os aplicativos de gerenciamento necessários que operam sobre o NOX.

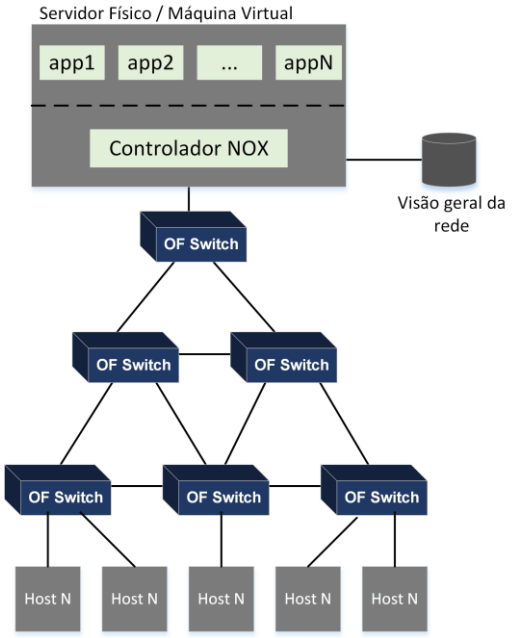


Figura 11 – Componentes de uma rede baseada em NOX.

Posteriormente ao NOX, que é baseado em *C++ e Python*, diversos outros controladores foram desenvolvidos nas mais variadas linguagens com suporte ao protocolo *OpenFlow*, dentre eles: POX [Kaur, S. et al., 2014], que é desenvolvido pela mesma equipe NOX e inteiramente baseado em *Python. Floodlight* [Floodlight, 2012], *Beacon* [Erickson, D., 2013], *OpenDaylight* [OpenDaylight, 2013], esses todos baseados em *Java*. A Tabela 2 faz uma breve relação entre as tecnologias citadas.

	NOX	**POX**	**Floodlight**	**Beacon**	**OpenDaylight**
Linguagem	C++	Python	Java	Java	Java
Suporte OF	Sim	Sim	Sim	Sim	Sim
Licença	GPLv3	GPLv3	Apache	GPLv2	EPL v1.0

Tabela 2 – Controladores OpenFlow - Características básicas.

A aplicação do modelo SDN sobre ambientes de nuvens computacionais necessita que o controlador estabelecido para gerenciamento da rede possua as características necessárias para operar neste cenário. Características como: operação de forma distribuída, integração com o protocolo *OpenFlow* e integração com sistemas gerenciadores de nuvens computacionais, são esperadas dos controladores.

2.3.3. Network Function Virtualization

O conceito de *Network Function Virtualization* (NFV) [B. Han,. Et al. 2015] [ETSI, 2012] surge como um complemento ao modelo SDN. A combinação de NFV com SDN tende a potencializar o desempenho, simplificar a compatibilidade entre tecnologias especialistas e facilitar procedimentos de manutenção de rede.

O conceito desperta interesse significativo da indústria e academia [Mijumbi, R., et al. 2016], através de sua importante mudança de paradigma sobre o provisionamento de serviços em telecomunicações. Essa mudança desacopla as funções de rede dos dispositivos físicos onde as funções rodam, o que pode ocasionar a redução de custos com despesas operacionais e de capital (*operating expenses* (OPEX) *e capital expenses* (CAPEX)), além de promover uma maior agilidade na implantação de novos serviços.

	SDN	**NFV**
Objetivo Geral	Separação do plano de dados do plano de controle, centralização e programabilidade da rede.	Migrar funções de rede de dispositivos dedicados para dispositivos genéricos.
Local de aplicação	Computação tradicional, Nuvens Computacionais.	Rede do provedor de serviço.
Dispositivos de aplicação	Servidores (*Commodity*) e switches.	Servidores (*Commodity*).
Padronização/Formalização	Open Networking Foundation (ONF).[3]	ETSI NFV Group.[4]
Protocolo	OpenFlow.	Não se aplica.

Tabela 3 – Quadro comparativo entre as tecnologias SDN e NFV.

Com o desacoplamento proposto é possível migrar as funções de rede implementadas por *hardwares* dedicados que executam funções específicas (e.g. Roteadores, *Firewalls*, *Gateways*, *Load Balancers*, NAT, etc.) para dispositivos genéricos (e.g. x86 comuns). Essa característica possibilita que as funções de rede virtualizadas sejam instaladas em qualquer ponto da rede, tornando o cenário totalmente dinâmico e adaptável às necessidades do consumidor de serviços ampliando as possibilidades do provedor. A Tabela 3 faz uma relação entre SDN e NFV.

[3] www.opennetworking.org/sdn-resources/sdn-definition

[4] www.etsi.org/technologies-clusters/technologies/nfv

2.4. Discussão do Capítulo

A era *Big Data* tem como principal motivador analisar uma grande quantidade de dados e transformá-los em informações no menor tempo possível, de forma a possibilitar tomadas de decisão ágeis e sobretudo assertivas. Conforme relatado na seção 2.1, variados *frameworks* para análise de processamento de *Big Data* foram propostos com diferentes abordagens, dentre elas o processamento em lote (*batch*) que requer a cópia de toda a demanda a ser executada e o processamento em tempo real (*stream*) que não necessita de armazenamento prévio e possui uma fonte infinita de dados.

A Tabela 4 relaciona as características elementares esperadas para cada tipo de metodologia, *batch* ou *stream*.

	Stream	***Batch***
Entrada	*Stream* desconhecido e interrupta de dados.	Conjunto fixo e finito de dados.
Armazenamento	Não necessita de armazenamento prévio dos dados. Em alguns casos necessita de uma parcela mínima em memória.	Necessita do armazenamento prévio dos dados.
Tempo de processamento	Milissegundos ou Segundos.	Minutos, Horas ou Dias.

Tabela 4 – Características elementares para processamento *batch* e *stream*.

Independente da abordagem executada pelo *framework*, essas ferramentas necessitam de uma infraestrutura robusta, escalável e segura para execução de suas demandas. A Computação em Nuvem apresenta os requisitos necessários para suportar as demandas computacionais geradas pelo modelo de processamento *Big Data*. Através da virtualização de *hardware* e compartilhamento de recursos providos pelo modelo *multi-tenant*, amplia as possibilidades do consumidor, que pode contratar a quantidade adequada de poder computacional para suas demandas e, se necessário, posteriormente aumentar ou reduzir a quantidade contratada de forma dinâmica.

A fim de oferecer ao modelo de nuvem, características em nível de rede como: programabilidade, configuração dinâmica baseada em requisitos definidos pelo administrador, centralização do plano de dados da rede, escalabilidade, flexibilidade e APIs com suporte a diferentes serviços, o modelo de SDN torna-se de extrema relevância para melhoria no desempenho.

O SDN atua de forma central no quesito infraestrutura de rede, abstraindo as camadas inferiores da rede e separando o plano de dados do plano de controle. Nesse sentido, fornecem maior flexibilidade de operação e transparência, características essas que ganham maior impacto quando combinadas com NFV, que tem como principal motivação migrar serviços que operam em *hardwares* dedicados para arquitetura de *hardware* comuns (e.g. x86).

Os modelos de Nuvem Computacional e SDN fornecem uma estrutura computacional de forma escalável e flexível. Nesse sentido, tornam-se adequadas para receber as altas e dinâmicas demandas geradas pelo modelo de processamento Big Data, operando conforme as necessidades de cada consumidor.

Capítulo 3

Trabalhos Relacionados

Esse capítulo se dedica a discutir os principais trabalhos relacionados aos seguintes temas: (i) Escalonamento de demandas em *Frameworks* para Big Data (lote e tempo real), (ii) Impacto do modelo *Multi-Tenant* e avaliação de desempenho de nuvens computacionais e (iii) Balanceamento de Carga nas Redes Definidas por Software - SDN.

3.1. Contexto Geral

Escalonar demandas computacionais de maneira a não sobrecarregar os nós de um ambiente é uma preocupação recorrente na literatura. O trabalho de [Singh, A., et al., 2008] implementa o algoritmo intitulado *VectorDot*, que objetiva manter sob controle a carga de trabalho incidente sobre os nós do sistema de um *Data Center*, de forma que quando o nó se encontra sobrecarregado o algoritmo realiza realocações (de VMs ou discos virtuais). Para isso, são utilizados dois vetores multidimensionais: (i) *NodeLoadFracVec(u)*, que representa fração de uso de cada recurso em relação ao servidor (*u*) e (ii) *ItemNodeLoadFracVec(vi, u) que* representa a carga da VM *(vi)* em relação ao servidor. Após calculados os valores dos referidos vetores, são ajustados os custos para migração e calculado o produto escalar entre esses vetores, o servidor físico que gerar o menor valor é escolhido para receber a VM.

No contexto de processamento *Big Data*, tipicamente, se faz necessário distribuir demandas em uma nuvem computacional de VMs já alocadas em nós físicos (VMs em execução). Isso desencadeou no aperfeiçoamento de soluções já aplicadas em *Data Centers (e.g VectorDot)*, que resultou na construção de inúmeras soluções que se propõem a avaliar os recursos computacionais no momento do escalonamento das tarefas. Tratar a interferência

multi-tenant (seção 2.2.2), característica do modelo em nuvem, é um tema desafiante presente na literatura.

3.2. Escalonamento para aplicações em Batch em Big Data

Esta seção relata dois trabalhos que têm como objetivo aprimorar o processo de escalonamento de demandas para o *framework Apache Hadoop* [Apache, 2016] através da implementação do modelo de programação *MapReduce* [Dean and Ghemawat, 2008]. Conforme mencionado na seção 2.1.1, *Hadoop* é originalmente projetado para processamento em *batch*. Os trabalhos discutidos na sequência são: *DyScale* [Yan, F., et al, 2015] e *Tetris* [Grandl, R., et al, 2014].

3.2.1. DyScale: a MapReduce Job Scheduler for Heterogeneous Multicore Processors

O trabalho de [Yan, F., et al, 2015] apresenta uma solução de escalonamento intitulada DyScale, que explora a capacidade de processamento oferecida por processadores *multi-core* para o escalonamento das demandas. Segundo o autor, não é claro o quanto uma aplicação *MapReduce* pode se beneficiar ao utilizar processadores com maiores frequências de processamento. Nesse sentido são executados testes em um cluster *Hadoop* que emprega a tecnologia de processadores *multi-core* que oferecem o controle sobre as frequências de CPU que variam de 1.6Ghz à 3.3Ghz. Dessa forma, é possível mensurar os resultados obtidos em diferentes frequências.

A solução *DyScale* altera a forma como a tarefa é instanciada no momento do escalonamento para que a tarefa seja alocada por afinidade de CPU. Dessa forma, uma tarefa pode ser definida a um núcleo ou a um conjunto de núcleos.

Para provar o problema evidenciado, os autores selecionaram 13 aplicações *MapReduce* disponíveis na literatura e executaram tais aplicações sobre um *cluster* composto por 8 nós, sendo 1 *master node* e 7 *slaves nodes*. Para provar o desafio destacado, as 13 aplicações elencadas foram executadas no *cluster* com duas configurações de frequência das

CPUs, sendo 1.6Ghz e 3.3Ghz respectivamente. Em todos os testes o tempo de execução das tarefas foi menor com a configuração 3.3Ghz.

Para as avaliações preliminares, a solução DyScale foi comparada com outras 3 soluções de escalonamento para o *framework Hadoop*: (i) FIFO (*First In First Out*), (ii) *Capacity Scheduler e Hadoop Fair Scheduler (HFS).* O ambiente de teste contou com 3 cenários que apresentam as seguintes características: Homogêneo Lento, Homogêneo Rápido e Heterogêneo. Para isso, os autores definem as configurações físicas de *hardwares* (CPUs com diferentes frequências) e suas devidas combinações. A solução mostrou que ao direcionar as aplicações para um *cluster* com um *pool* de recursos de *hardware* mais potente é possível melhorar o desempenho da aplicação em comparação com o escalonador padrão do *Hadoop* (40% de ganho no cenário Homogêneo Rápido), e também em determinados cenários quando comparado com *Fair Scheduler* (30% Heterogêneo) e *Capacity Scheduler* (30% Heterogêneo).

A solução DyScale promoveu um entendimento, esse já previsto, que ao se avaliar o poder computacional real dos processadores e alocar tarefas para executar sobre esses existe um ganho relativo de desempenho em relação a escalonadores que são baseados somente no padrão FIFO. Um fator complicador na solução apresentada é que existe a necessidade de conhecer as configurações de *hardware* dos computadores para aplicar a solução e alteração das tarefas do *Hadoop*.

3.2.2. Multi-Resource Packing for Cluster Schedulers

O trabalho de [Grandl, R., et al, 2014], intitulado Tetris, é uma iniciativa da Microsoft em parceria com pesquisadores da Universidade de *Wisconsin, Madison*. Trata-se de um escalonador para *clusters Hadoop* que considera quais recursos as tarefas necessitam (e.g. CPU, Disco, Memória ou Rede) e realiza a distribuição das tarefas baseando-se nessas necessidades. Segundo os autores, a maioria das soluções de escalonamento para os *frameworks* de Big Data tipicamente consideram apenas *Slots* disponíveis baseados na quantidade de memória [Isard, M., et al, 2009], [C.S., 2016], [F.S., 2016], [YARN, 2016] e dessa forma a memória torna-se o único recurso físico (real) avaliado. Por sua vez, não consideram recursos adicionais como disco e rede, que são de extrema necessidade para aplicações de *Big Data*. Assim, quando diferentes tarefas necessitam de um recurso específico, como por exemplo rede, serão escalonadas ao

mesmo tempo e isso pode ocasionar uma utilização demasiada do recurso em questão impactando no desempenho das tarefas escalonadas.

No Tetris, o nó central (*master*) analisa os recursos disponíveis e atribui as tarefas conforme a disponibilidade do *cluster*. Para isso, cada nó escravo (*slave*) realiza a leitura dos recursos disponíveis e informa de forma periódica seu estado físico (real) ao *master node*.

Outro ponto considerado para o escalonamento na solução em questão é a alteração e incorporação de múltiplos recursos no algoritmo *Smallest Remaining Time First* (SRTF), que realiza o escalonamento de tarefas menores primeiro, segundo os autores, e isso impacta de maneira positiva no processamento das tarefas. Para o Tetris realizar o escalonamento das tarefas é necessário conhecer a aplicação para distribui-las com base nos recursos necessários para a tarefa e também nos recursos disponíveis no *cluster*. Tarefas que utilizam o mesmo recurso (recorrentes) são enviadas para o mesmo destino.

Para avaliação da solução, os autores utilizaram um cluster com 250 nós, estes cada um com 32 núcleos, 96GB de memória, disco de 8 TeraBytes, rede 10 Gbps com S.O. Linux 2.6.32, como *baseline.* Os autores executaram os testes com escalonadores padrões da literatura: *Capacity Scheduler* [C.S., 2016] e *Fair Scheduler* [F.S., 2016]. A carga avaliada é baseada na reprodução dos *logs* gerados pela rede social *facebook*, que imitam a carga real e contemplam aspectos como: recursos necessários para cada *Job*, tamanhos de entrada e saída, e localização desses arquivos no HDFS.

As avaliações preliminares relacionadas ao tempo de conclusão das tarefas apresentaram que o Tetris é capaz de finalizar o processamento dos *Jobs* em média 40% mais rápido que as soluções tradicionais. Considerando o tempo de finalização do conjunto completo dos *Jobs*, o Tetris torna-se 33% mais rápido que os escalonadores tradicionais.

Dentre os escalonadores para *frameworks* de *Big Data* para *batch*, a solução Tetris apresenta uma estratégia que considera diferentes recursos computacionais, avaliando rede, CPU, memória e disco. De forma adicional, analisam quais recursos as tarefas necessitam para um escalonamento inteligente e direcionado. Outro fator que está incluso na estratégia de escalonamento é a prioridade atribuída a tarefas menores.

3.2.3. Discussão da seção

Os trabalhos apresentados nessa seção realizam o escalonamento de demandas para o *framework* Apache Hadoop. Conforme já mencionado na seção 3.2.2, o processo de escalonamento do Hadoop, em sua forma nativa, considera apenas a quantidade de memória e *slots* disponíveis em cada nó escravo. Partindo dessa propriedade, os trabalhos relatados tratam o processo de escalonamento considerando recursos adicionais, conforme sumarizado na Tabela 5.

Solução	**Recursos Virtuais Considerados (S.O.)**	**Recursos Fisícos (Reais) Considerados (*Hardware*)**	**Trata *MultiTenant***	**Requer conhecimento prévio do tipo da tarefa**
DyScale	CPU	-	Não	Sim
Tetris	CPU e Memória	Disco e Rede	Não	Sim

Tabela 5 – Comparativo soluções de escalonamento para o Framework Apache Hadoop

A solução DyScale considera o poder computacional e apresenta que o escalonamento de tarefas para processadores mais rápidos permite um ganho de desempenho no processamento das demandas.

No caso do Tetris, a solução considera os recursos CPU, memória, rede e disco. Porém, é necessário o conhecimento de quais recursos a aplicação necessitará e a partir disso definir o processo de escalonamento baseado nos dados informados pelos agentes que operam nos nós escravos. Dessa forma, a solução não é transparente para o usuário.

Embora ambas a soluções tratem de forma diferenciada o processo de escalonamento e considerem recursos computacionais que o escalonador padrão *Hadoop* não considera, nenhuma solução apresenta técnicas que considerem o desafio *Multi-Tenant*.

3.3. Escalonamento para aplicações de *Data Stream* em *Big Data*

Esta seção relata alguns trabalhos que tem como objetivo aprimorar o processo de escalonamento de demandas do *framework Apache Storm* (subseção 2.1.2), uma ferramenta para processamento de *stream* (tempo real). Os trabalhos são: (i) *Adaptive Scheduling* [Aniello, L., et al, 2013], (ii) *T-Storm* [Xu, J., et al, 2014], (iii) *R-Storm* [Peng, B., et al, 2015].

3.3.1. Adaptive Scheduling

No quesito escalonamento para o *framework Apache Storm*, de forma pioneira na literatura, foi desenvolvido o *Adaptive Scheduling* [Aniello, L., et al, 2013]. Seu principal objetivo é considerar a comunicação entre os *executors* (discutidos na seção 2.1.2) e escalonálos preferencialmente no mesmo *slot* e consequentemente no mesmo nó. Para isso, foram propostas duas abordagens de escalonamento: *offline scheduler* e *online scheduler*.

A política *offline scheduler* é baseada estritamente na topologia, onde o escalonador define quais *slots* são mais convenientes para realizar o processamento da demanda. Esse procedimento ocorre de forma prévia ao escalonamento e não considera outras variáveis como CPU ou memória, por exemplo. A principal heurística aplicada nesse procedimento é avaliar quais *executors* possuem comunicação direta (e.g. $bolt_i$ consome as tuplas geradas pelo $bolt_j$) e ajustar o escalamento para alocá-los no mesmo *slot* de forma a evitar as latências de rede.

Já a abordagem *online scheduler* verifica no momento do escalonamento o número total de *executors* da topologia, número de *slots* disponíveis em cada *worker node* e adicionalmente o poder computacional virtual de cada *worker node*. Para computar o poder computacional disponível nos *worker nodes* de destino, foi utilizado a API *Java* que contém um método chamado *getThreadCpuTime (threadID)* que permite avaliar o status dos nós do *cluster* e detectar *worker nodes* com sobrecarga no momento do escalonamento. Os dados relacionados ao estado computacional são coletados através de agentes que operam nos *worker nodes*. Essas informações são gravadas em um banco de dados centralizado que é consultado pelo escalonador a cada interação.

Os testes preliminares do *Adaptive Scheduling* avaliaram o tempo médio de processamento por tupla, apresentando melhor desempenho nas duas abordagens *online* e *offline*, quando comparado ao *EvenScheduler* (escalonador do *Storm*). Dessa forma, a política *online* obteve resultados superiores com ganhos de 20% a 30% na velocidade do processamento quando comparado à política padrão. No caso do *offline scheduler*, os próprios autores consideram a abordagem ineficiente na prática, por não considerar as cargas computacionais dos *worker nodes*. Quanto ao *online scheduler*, uma limitação encontrada e destacada em [Xu, J., et al, 2014] é a necessidade de instrumentação do código fonte da topologia para permitir o

monitoramento da carga computacional e fornecer ao *online scheduler*. Portanto, essa solução *Adaptive Scheduler* não é transparente para o usuário do *Storm*.

3.3.2. T-Storm

O trabalho intitulado *T-Storm* relatado em [Xu, J., et al, 2014] apresenta uma solução de escalonamento que tem como objetivo considerar as cargas computacionais para escalonar e reescalonar tarefas. A solução considera como prioridade a comunicação inter-nó e interprocesso, alocando tarefas no menor número de nós para reduzir inter-tráfego entre *executors*. Desta maneira, segundo os autores, é possível obter melhor desempenho sobre as aplicações com um número reduzido de *worker nodes*.

De forma semelhante *ao Adaptive Scheduler*, *T-Storm* utiliza agentes instalados nos *worker nodes* que executam a API *Java* que contém o método *getThreadCpuTime(threadID)* para avaliar o status da CPU e detectar sobrecargas nos *worker nodes*. Para identificar o tráfego entre *executors*, foi realizada uma alteração no código-fonte do *Storm* para avaliar o número de tuplas enviadas entre pares de *executors*, durante o período de 20 segundos, pois, segundo os autores, assim é possível avaliar quais *executors* possuem altas taxas de comunicabilidade e assim pode-se realizar o reescalonamento para que operem no mesmo *worker node*, minimizando as latências dessa comunicação. Os agentes responsáveis por coletar as taxas relacionadas à carga de trabalho registram as informações coletadas em um banco de dados centralizado, que é utilizado pelo *T-Storm* no momento do escalonamento e reescalonamento.

Para comparar o desempenho com o *EvenScheduler*, avaliações preliminares foram realizadas utilizando-se de topologias padrões disponíveis na literatura como *Word Count topology* e *Throughput Test topology*. Os testes revelaram que *T-Storm* apresenta um desempenho superior na execução das tarefas em relação ao *EvenScheduler*, com ganhos de 27% na velocidade do processamento. Isso se justifica pela aplicação de políticas de escalonamento que consideram as variáveis adicionais, como o poder computacional do *worker node* e a comunicação *inter-exercutors*.

3.3.3. R-*Storm*

A abordagem de escalonamento proposta em [Peng, B., et al, 2015], intitulada R-Storm, considera os seguintes recursos computacionais: CPU, memória e rede. A solução é composta de 3 módulos: (i) *StatisticServer*: módulo que coleta informações referentes à estatística do *Cluster Storm*, que podem ser: vazão de uma topologia, componentes e nível da topologia. (ii) *GlobalState:* Módulo principal da solução que armazena as informações sobre a localização das tarefas no *cluster*, informações sobre os recursos físicos (reais) das máquinas que compõem o *cluster Storm*, informações sobre a demanda escalonada e o que ainda necessita ser escalonado. (iii) *ResourceAwareScheduler:* módulo que implementa a interface *IScheduler* do *Storm* e contém a implementação do *R-Storm*.

Para cada topologia, o *R-Storm* fornece uma API em que o usuário informa a quantidade de recurso que essa aplicação necessita (memória e CPU). Para a carga computacional disponível no *worker node*, sugere-se alterar os parâmetros do arquivo de configuração do *Cluster Storm* (localizado em *conf/yaml.storm*) e informar a capacidade computacional de CPU e memória disponível para cada nó. Para emular a latência de rede *inter-rack*, a solução utiliza de duas VLANs, cada uma com 6 *worker nodes*, totalizando 12 nós, e aplica algoritmo *round-trip-time*, para assim a solução mapear os nós que fazem parte da mesma VLAN e reduzir a latência de rede ao alocar os executores na mesma VLAN.

As avaliações preliminares realizaram a execução das topologias desenvolvidas pelos autores em relação ao *EvenScheduler*. Os experimentos concentraram-se em duas situações: *Network Resource Bound* para testar o comportamento de topologia que faz o uso de rede e *Computation Time Bound* para topologias que fazem o uso de CPU. O desempenho do *R-Storm*, se mostrou superior ao escalonador padrão em todos os testes apresentados, apresentando melhoras de 47% para aplicações de rede e 69% para aplicações de CPU.

3.3.4. Discussão da seção

Essa seção apresentou soluções de escalonamento inerentes ao *Apache Storm*. Todos os trabalhos discutidos evidenciaram a necessidade do aprimoramento das técnicas de escalonamento no contexto de processamento *Stream* para Big Data, pois, se somente forem

avaliados o número de *slots* e quantidade de *executors*, características exclusivas do *framework* em questão, o resultado pode ser ineficiente. Uma vez que o poder computacional de um nó físico (*worker node*) pode encontrar-se comprometido e, ao alocar tarefas a esse nó, o desempenho pode ser reduzido de forma significativa.

As soluções *Adaptive Scheduler* e *T-Storm* utilizam de um banco de dados centralizado para armazenar informações que são coletadas por agentes que atuam nos *worker nodes*. Essas soluções avaliam o consumo de CPU baseado nas estatísticas do sistema operacional, e o atualizam para que a solução de escalonamento aplique a sua política. Avaliar somente as taxas do sistema operacional não garante que o *hardware* esteja ou não comprometido, pois, em ambientes com *hardware* virtualizado, mais de uma VM podem coexistir sobre um mesmo *hardware* físico. A principal diferença entre essas soluções é que o *T-Storm* apresenta uma solução transparente ao usuário, uma vez que o *Adaptive Scheduler* necessita de alterações no código fonte da topologia para atuar no *Cluster Storm*.

A solução apresentada em *R-Storm* é semelhante ao *Adaptive Scheduler*, e demanda intervenção por parte do usuário. É necessário informar de forma manual a quantidade de recursos computacionais (memória e CPU) que cada aplicação demanda para sua execução. Para isso, o *R-Storm* disponibiliza uma API para o usuário informar tais valores. Quanto ao poder computacional disponível pelo *worker node*, é sugerida alteração no arquivo de configuração central do *Cluster Storm*, o que torna a solução não transparente ao usuário e as configurações estáticas para cada topologia em execução. Essa abordagem exige que o usuário possua conhecimento prévio sobre o ambiente físico e sobre a aplicação que deseja processar.

A Tabela 6 sumariza o que cada solução apresentada nessa seção aborda para o escalonamento das demandas computacionais.

Solução	**Recursos Virtualizados Considerados (S.O.)**	**Recursos Físicos Considerados (*Hardware*)**	**Trata *Multi-Tenant***	**Requer alteração no código fonte da aplicação**
Adaptive Scheduler	CPU	-	Não	Sim
T-Storm	CPU	-	Não	Não
R-Storm	CPU e Memória	Rede	Não	Sim

Tabela 6 – Comparativo de soluções de escalonamento para o Framework Apache Storm

Todas soluções relatadas nessa seção não consideram o desafio *multi-tenant* para o escalonamento das tarefas. Como já mencionado anteriormente (seção 2.1), aplicações para processamento de *Big Data* necessitam de uma infraestrutura robusta para execução de suas

tarefas. Tal característica é encontrada nos ambientes de nuvem computacional, que por sua vez utilizam o modelo *multi-tenant* para a virtualização do *hardware*. Por isso, é de extrema importância, para o melhor desempenho das aplicações, considerar o poder computacional físico (real) e virtual dos nós encarregados do processamento das tarefas.

3.4. Impacto do modelo *Multi-Tenant* sobre Nuvens Computacionais

Conforme mencionado na fundamentação, o modelo de computação em nuvem possui características como: alta escalabilidade, virtualização de recursos computacionais, facilidade de operação por parte do consumidor e infraestrutura robusta. Tais propriedades do modelo desencadearam uma ampla utilização para os mais distintos fins, dentre eles o processamento de Big Data.

Um desafio relacionado ao modelo de nuvem computacional que é abordado na literatura, é impacto que o modelo *multi-tenant* pode ocasionar sobre as demandas de processamento dos consumidores dos serviços da nuvem. Nesse sentido, essa seção apresenta dois estudos que detectam e evidenciam a necessidade de técnicas que avaliem esse cenário.

A seguir, os seguintes trabalhos serão discutidos: *Runtime Measurements in the Cloud* [Schad, J. et al, 2010], *Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies* [Hwang, K., et al, 2015].

3.4.1. Runtime Measurements in the Cloud

O estudo de [Schad, J. et al, 2010] realiza uma avaliação exaustiva na plataforma Amazon EC2[5] que é amplamente utilizada em plataformas de nuvem computacional. Os experimentos se subdividem em três fases: (i) instância única EC2 para estimar a variação de desempenho de um nó virtual executado individualmente. (ii) múltiplas instâncias EC2 (cinquenta nós virtuais) para estimar a variação de desempenho com vários nós virtuais. (iii)

[5] aws.amazon.com

diferentes localizações (Estados Unidos e Europa) para estimar a variação de desempenho em *clusters* localidades geográficas distintas.

A metodologia de testes engloba uma série de quesitos, conforme os autores elencam, como os principais componentes que são avaliados em uma nuvem que podem provocar um impacto significativo de desempenho: (i) Instância de inicialização: que mede o tempo entre a realização de pedido de uma instância e o tempo que a demanda necessita para ser atendida. (ii) CPU: componente crucial para maioria das aplicações. (iii) Memória: também componente crucial para inúmeras aplicações. (iv) Disco: componente chave para aplicações de nuvem, que necessitam armazenar seus dados de forma prévia ao processamento. (v) Largura de banda entre instâncias: aplicações que executam em ambientes de nuvem processam uma grande quantidade de dados e necessitam enviar/trocar dados através da rede. (vi) Acesso externo ao S3 (*Simple Storage Service*): avaliar a taxa de *upload* é importante, pois a maioria dos usuários inicialmente realiza o *upload* dos arquivos para S3, e posteriormente realiza o processamento. Para medir o desempenho de cada componente foram utilizadas ferramentas de *benchmark* de mercado, como *ubench*[6] para CPU e Memória, *Boniie++*[7] para disco e *Iperf*[8] para rede.

As avaliações foram executadas durante 31 dias, duas vezes por hora, e a cada hora foram realizados *benchmarks* para cada instância, esses sincronizados para evitar sobreposição de *benchmarks* entre instâncias. Ao final da primeira hora, todos as instâncias eram desligadas para iniciar outro teste, pois, segundo os autores, ao solicitar nova instância de uma VM para a nuvem, essa pode ser alocada em local fisicamente diferente da instância anterior e assim é possível ter uma visão mais ampla do processo como um todo. Para comparar os resultados, os autores executaram os mesmos experimentos em um *cluster* local, com exceção do S3. Por se tratar de um ambiente controlado pelos autores, esse representa a *Baseline* para os experimentos.

---

[6] phystech.com/download/ubench.html

[7] textuality.com/bonnie/intro.html

[8] iperf.sourceforge.net/

Para avaliar o impacto real sobre o desempenho de uma aplicação, o *Hadoop* foi utilizado nas avaliações. Sua escolha se dá por ser uma tecnologia que faz o uso intenso dos componentes de disco, rede e CPU. Essa ferramenta realizou operações matemáticas sobre diferentes cargas de trabalho, de 25GB e 100GB. Para todos os resultados, os autores identificaram um coeficiente de variação de 11% entre os resultados, o que pode causar impactos significativos em experimentos que necessitam de repetibilidade. Para os *benchmarks* que avaliaram a variação relacionada aos recursos computacionais, os coeficientes de variação apresentaram os seguintes valores: 24% para desempenho de CPU, 20% para o desempenho de disco e 19% para o desempenho da rede. Para a avaliação do S3, o coeficiente de variação foi de 53%.

Os resultados destacados nesse trabalho expressam a importância de se avaliar o poder computacional do *hardware*, pois, durante as execuções dos testes, ocorreram variações que impactam de forma significativa o desempenho das aplicações.

3.4.2. Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies

O estudo de [Hwang, K., et al, 2015] avalia diversas métricas de desempenho sobre a plataforma de nuvem Amazon EC2: (i) Velocidade (ii) Eficiência, (iii) Utilização e (iv) Escalabilidade. Para essa avaliação foram utilizadas cinco ferramentas de *benchmark* para nuvem: *BenchCloud, CloudSuite, HI Bench, TPC-W, YCSB.*

O trabalho destaca que nuvens computacionais dependem de técnicas de virtualização para garantir a elasticidade, e nesse sentido o modelo *multi-tenant* utilizado nas nuvens computacionais proporcionam suporte ao processamento de inúmeras tarefas.

Diferentemente do estudo anterior [Schad, J. et al, 2010] em que os autores replicaram os testes em uma nuvem local, os experimentos de [Hwang, K., et al, 2015] avaliam uma aplicação executando em diferentes quantidades de *hosts* instanciados na nuvem conforme a rotina de teste. Os testes evidenciam que a nuvem apresenta uma variação de desempenho em diversos momentos.

Um dos testes realizados verifica o processamento de uma carga de trabalho utilizando a aplicação *Word Count* através do *Hadoop*. Para esse teste, os autores utilizaram arquivos com os seguintes tamanhos: 10GB, 5GB e 1GB. O número de instâncias para o teste variou da seguinte forma: 1, 2, 4, 8, 12 e 16 *hosts* virtuais. Para todos os tamanhos de dados, o desempenho (medido em MB/segundo) sofreu variações. Por exemplo, o arquivo de 1GB que foi executado via quatro instâncias alcançou a taxa de aproximadamente 25 MB/s. Quando foi utilizado 16 instâncias, a taxa foi de aproximadamente 10MB/s, representando uma variação de aproximadamente 40%.

Os autores realizaram outros testes, comparando as ferramentas de *benchmark* de nuvem entre si e comparando os resultados obtidos entre as cinco ferramentas. De maneira geral, os resultados apresentados destacam a necessidade da utilização de ferramentas que avaliem, através de *benchmarks*, o estado nuvens computacionais.

3.4.3. Discussão da seção

Essa seção apresentou dois estudos (Tabela 7) que evidenciaram a variação de desempenho que aplicações podem sofrer ao serem executadas em uma nuvem computacional. Para isso, os trabalhos utilizaram o *Hadoop* como solução de mercado para provar a variação que aplicações reais podem sofrer nesses ambientes. De forma adicional, os autores realizaram avaliações com ferramentas de mercado que utilizam técnicas de *benchmark* para avaliar o estado físico dos nós que estão em instanciados na nuvem, o que permitiu a visualização de variações inerentes ao cenário de nuvem computacional.

Solução	**Detecta variação no desempenho de aplicações**	**Aplicação utilizada para avaliação**	**Detecta variação desempenho do hardware**	**Técnica utilizada para detectar variação no hardware**
Runtime Measurements in the Cloud	Sim	Aplicação *MapReduce*	Sim	*Benchmarks*
Cloud Performance	Sim	Aplicação *MapReduce*	Sim	*Benchmarks*

Tabela 7 – Comparativo de técnicas para avaliação de desempenho de nuvens computacionais

3.5. Redes Definidas por Software para Balanceamento de Carga

De maneira geral, a estratégia de balanceamento de carga é aplicada em grandes infraestruturas computacionais. Essas infraestruturas tipicamente constituem-se de múltiplos servidores replicados que oferecem o mesmo tipo de serviço e encontram-se conectados através de uma rede de dispositivos de encaminhamento (*switches*). Cada réplica de servidor possui um endereço IP único e os clientes acessam aos serviços através de um endereço de IP público, esse também único, que opera como o *gateway* dos servidores. O *Load Balancer* (seção 3.5) opera reescrevendo o endereço de destino a cada solicitação dos clientes, conforme a política de balanceamento definida. Seu objetivo é realizar melhor a distribuição das tarefas dentre os servidores disponíveis nos grandes centros de dados.

Segundo [Kreutz, D., et al, 2015], o *Load Balancer* foi uma das primeiras aplicações implementadas e testadas no modelo SDN. Nesse sentido, esta seção apresenta algumas propostas que aplicam o balanceamento de carga sobre o modelo SDN. Os trabalhos discutidos são: *Plug-n-Serve: Load-BalancingWeb Traffic using OpenFlow* [Handigol, N., et al, 2009], *Aster * x: Load-Balancing Web Traffic over Wide-Area Networks* [Handigol, N., et al, 2010] e *OpenFlow-based server load balancing gone wild* [Wang, R., et al, 2011].

3.5.1. Plug-n-Serve: Load-BalancingWeb Traffic using OpenFlow

Plug-n-Serve [Handigol, N., et al, 2009] é uma solução para balanceamento de carga lançada por membros da mesma equipe que propôs o modelo SDN e protocolo *Openflow*, da universidade de *Stanford*. A solução propõe implantar e avaliar o comportamento do *Plug-n-Serve* no campus da universidade, com objetivo de minimizar o tempo de resposta para solicitações Web. Para isso a solução avalia os enlaces de rede e a carga de CPU de servidores da rede de forma a escolher rotas com menores congestionamentos e servidores com maior poder computacional de CPU disponível.

Todos os clientes realizam requisições para um mesmo IP, e essas requisições são reescritas pelo balanceador de carga com base em suas políticas. Para o funcionamento da solução, foram propostos 3 módulos: (i) *Flow Manager*: opera como um controlador que gerencia rotas de fluxos baseados no algoritmo de balanceamento de carga escolhido para

operação, nesse módulo que o algoritmo de balanceamento de carga deve ser implementado. (ii) *Net Manager*: Módulo que monitora o estado da rede, de forma a avaliar o estado de utilização da rede como um todo. (iii) *Host Manager*: Monitora o estado individual da carga dos servidores, sendo responsável por detectar novos servidores.

O documento não apresenta avaliações preliminares, mas relata que a solução foi empregada e opera em ambiente de produção na universidade que contém dezenas de servidores *web* espalhados em uma rede *OpenFlow*. Essa rede opera em conjunto com uma coletânea heterogênea de modelos de *switches* de diferentes marcas (e.g.: Cisco, HP e NEC). Os *switches OpenFlow* são controlados remotamente pela solução *Plug-n-Serve* que atua em um computador separado rodando sobre o NOX.

3.5.2. Aster * x : Load-Balancing Web Traffic over Wide-Area Networks

O trabalho de [Handigol, N., et al, 2010] é uma evolução do *Plug-n-Serve* (seção 3.5.1) e apresenta uma solução de balanceamento de carga intitulada *Aster*x*. Essa solução apresenta um controlador que opera em conjunto com o protocolo *Openflow* e objetiva minimizar o tempo médio de resposta em redes não estruturadas. Seu diferencial em relação ao *Plug-n-Serve* é que, na medida que novos recursos computacionais e *switches* são adicionados à rede, a solução *Aster*x* expande de forma automática sua visão da rede, adicionando essas novas entidades para realizar o balanceamento de carga de forma adequada.

Todos os servidores respondem a um mesmo IP quando uma solicitação nova ocorre, no qual é encaminhada para um servidor específico. Para isso, o *Aster*x* determina o estado atual da rede e dos servidores, incluindo topologia da rede, tráfego, congestionamento e a carga dos servidores. Os módulos para o funcionamento da solução são os mesmos presentes na proposta precursora (*Plug-n-Serve*): (i) *Flow Manager*. (ii) *Net Manager* (iii) *Host Manager*, todas já relatadas na secção 3.5.1.

Esse trabalho não realizou avaliações preliminares e deixou a abordagem Aster*x como principal contribuição de balanceamento de carga em nível de WAN (*Wide Area Networks*), que tem como objetivo reduzir o tempo de resposta para aplicações *Web* pelo fato de escolher rotas pouco congestionadas e servidores com maior poder de processamento disponível. Para aplicar essa proposta, os autores pretendem implantar a solução *Aster*x* sobre

uma fatia da rede GENI (*Global Environment for Network Innovations*) [GENI, 2006], que é uma iniciativa que oferece um laboratório virtual para redes e sistemas distribuídos para pesquisadores realizarem avaliações de suas soluções. Seu principal diferencial em relação à solução precursora (*Plug-n-Serve*) é a capacidade de ampliar a visão da rede e detectar novos nós e dispositivos de encaminhamento de forma automática.

3.5.3. OpenFlow-based server load balancing gone wild

A proposta de [Wang, R., et al, 2011] utiliza o modelo SDN e o protocolo *Openflow* para realizar o balanceamento de carga e a solução é pautada na utilização de regras coringas (*wildcard rules*) que são implantadas nos *switches* SDN, conforme as alterações de cargas dos servidores e as regras anteriores são removidas. Dessa forma, o novo tráfego será direcionado para o servidor réplica com recurso em maior abundância, a solução que garante que conexões já existentes e em operação não sejam descontinuadas, e assim a regra fica ativa para tráfegos específicos e, posteriormente, ao fim da conexão, é removida automaticamente.

As avaliações preliminares tiveram como objetivo testar a solução sobre uma arquitetura com 36 clientes que solicitavam um arquivo (*wget requests*) de forma aleatória, com intervalos variando de 0 a 10 segundos para 3 réplicas de servidores. A lógica para redirecionamento de fluxo foi pré-fixada pelos autores de forma que cada servidor recebe uma porcentagem do tráfego e dessa forma não existe política para avaliar a carga propriamente dita, sendo avaliado somente o comportamento sobre as inclusões/alterações/remoções de regras coringas para tratar os novos fluxos. Dessa forma, os testes revelaram que a solução não gera impactos significativos de desempenho no controlador. Não houve comparação com outra técnica de balanceamento de carga.

3.5.4. HAVEN: Holistic Load Balancing and Auto Scaling in the Cloud

O trabalho de [Poddar, R., et al, 2015] apresenta um sistema de balanceamento de carga que opera em nuvem computacional, intitulado *Haven*. A solução monitora os níveis de congestionamento dos links de rede e coleta as estatísticas de utilização de recursos em nuvem, mais especificamente CPU e memória das máquinas virtuais em operação em seu cluster, que

são solicitados ao respectivo hipervisor. Através das medidas citadas é calculado um escore, e posteriormente seleciona-se o destino da demanda.

Os autores afirmam que o trabalho é considerado como uma melhoria da *solução Plug-n-Server* (seção 3.5.1), principalmente pelo fato de utilizar a arquitetura *OpenStack*[9], o que melhora a escalabilidade da solução de maneira geral e permite a integração de um balanceador de carga customizado. *Haven*, faz uso da tecnologia das Redes Definidas por *Software* para garantir que as políticas de balanceamento de carga personalizadas sejam executadas no plano de controle, de maneira que o restante do tráfego permaneça no plano de dados.

3.5.5. Discussão da seção

Essa seção discutiu soluções de balanceamento de carga que utilizam SDN para a distribuição de demandas computacionais dentre os servidores disponíveis para processamento. A Tabela 8 faz uma relação entre as soluções.

As soluções *Plug-n-Server* e *Aster*x*, são propostas por membros que pertencem ao grupo que desenvolveu o SDN, da universidade de *Stanford.* O objetivo das duas soluções é o mesmo: através da técnica de balanceamento de carga, minimizar o tempo de resposta das requisições realizadas aos servidores, através da avaliação dos *links* de comunicação (congestionamento) e avaliação das cargas de CPU do servidor. As duas propostas não apresentaram avaliações preliminares. No caso do *Plug-n-Serve*, os autores afirmam que a solução está em operação no campus universitário, já a *Aster*x* é uma proposta de expansão do *Plug-n-Serve* para operar em ambientes mais robustos, no caso uma WAN, e incluir de forma dinâmica novos dispositivos de encaminhamento e servidores no plano de controle para que *Aster*x* realize o balanceamento.

[9] https://www.openstack.org/

Solução	Trata Fluxo de Rede	Trata Carga do Servidor (Lógico)	Trata Carga do Servidor (Físico)	Operam em nuvem computacional
Plug-n-Serve	Sim	Sim (CPU)	Não	Não
Aster*x	Sim	Sim (CPU)	Não	Não
Load Balancing Gone Wild	Não	Não	Não	Não
HAVEN	Sim	Sim (CPU e Memória)	Não	Sim

Tabela 8 – Comparativo de soluções de Balanceamento de Carga sobre SDN

A na solução *Load Balancing Gone Wild*, tem como principal contribuição a criação de regras coringa (*wildcard rules*) e sua avaliação se concentra no quesito de que, no instante da alteração/substituição de regras coringa, conexões já ativas entre clientes e servidores não sejam finalizadas, sendo que ao fim dessa conexão a regra coringa (antiga) é eliminada.

Finalmente, a solução intitulada *Haven* apresenta uma proposta de balanceamento de carga em nuvem computacional que utiliza o *framework OpenStack* como estudo de caso e faz a avaliação dos estados virtuais (CPU e memória) dos nós atrelados à taxa dos *links* de comunicação para definir em qual nó destino a demanda será processada.

Através dos trabalhos apresentados nessa seção, reitera-se a necessidade de proposta e desenvolvimento de um método de balanceamento de carga que opere no cenário de nuvens computacionais de maneira a avaliar um conjunto de variáveis (e.g. fluxo de rede, poder computacional virtual e poder computacional físico do conjunto de nós disponíveis) para possibilitar tomadas de decisão mais precisas no momento do balanceamento das demandas.

3.6. Discussão do Capítulo

Esse capítulo dissertou sobre os principais trabalhos relacionados dentro dos seguintes temas: (i) escalonamento de demandas para *frameworks* para Big Data, (ii) avaliação do estado físico de nuvens computacionais, e (iii) Redes Definidas por Software (SDN) para balanceamento de carga. Tais trabalhos fornecem subsídios para a atual proposta de pesquisa (descrita no Capítulo 4).

Inicialmente, nas seções 3.2 e 3.3, foram apresentadas soluções de escalonamento para *frameworks* de *Big Data* para processamento em lote (*batch*) e em tempo real (*stream*). As soluções apresentaram como principal objetivo o desenvolvimento de políticas de escalonamento que consideram variáveis adicionais (e.g.: poder computacional, tipo do recurso

que aplicação necessita etc.) no momento do escalonamento das respectivas demandas computacionais. A principal limitação encontrada nesses trabalhos foi a ausência de políticas que tratem o desafio do compartilhamento de recursos de *hardware* (*multi-tenant*). Ao propor soluções de escalonamento para *frameworks* de Big Data, quando se almeja aprimorar o desempenho das aplicações, torna-se de extrema relevância considerar o estado físico (real) do *hardware*, pois, de forma típica, *frameworks* de Big Data rodam em ambientes de nuvem computacional pela necessidade de infraestrutura adequada para o seu processamento.

A seção 3.4 apresentou dois trabalhos que constatam as variações de desempenho que ocorrem em ambientes de nuvem computacional. Os autores executam a mesma aplicação de forma exaustiva em momentos distintos e obtiveram resultados com variações significativas. De forma adicional, empregaram ferramentas que utilizaram técnicas de *benchmark* para avaliar o estado físico do *hardware* e igualmente constataram grandes variações nos resultados fornecidos por essas ferramentas. Desse modo, esses trabalhos relatam que utilizar técnicas de *benchmark* para avaliar o estado do *hardware* físico pode possibilitar a coleta de informações reais do poder computacional no qual uma VM está alocada.

Por fim, a seção 3.5 apresentou 4 soluções que aplicam o SDN no balanceamento de carga entre servidores. As soluções apresentadas não tratam a questão de recurso compartilhado, avaliando somente o estado virtual dos nós em questão, o que não é a melhor estratégia conforme as avaliações realizadas no Capítulo 5.

Capítulo 4

Proposta

Esse capítulo apresenta a proposta desta pesquisa.

4.1. Contexto Geral

Esta pesquisa apresenta 3 soluções, que de forma conjunta objetivam aprimorar o desempenho de aplicações baseadas em *data stream* para *Big Data*, executadas em ambiente *multi-tenant* em nuvem computacional. As soluções são: (i) escalonador e reescalonador de demandas *data stream* para *Big Data*, que operam sobre um conjunto de nós responsáveis pelo processamento das demandas, chamado ***Dynamic Scheduler (DySc)***; (ii) mecanismo de provisionamento de recursos baseado em NFV, chamado ***Elastic Resource Provisioning (ERPr)***; (iii) balanceador de carga para distribuição de demandas computacionais entre múltiplos *clusters multi-tenant*, chamado ***Dynamic Load Balancer (DyLB)***.

DySc é responsável pelo escalonamento e, posteriormente, pelo reescalonamento periódico das demandas de *data streaming* entre nós com recursos físicos disponíveis dentro de uma infraestrutura de nuvem (*cluster*[10]) específica. A avaliação dos recursos físicos é um diferencial do trabalho em relação aos escalonadores propostos na literatura (seção 3.3), que operam somente baseados em recursos virtuais. Por outro lado, o ERPr é responsável pela detecção, de forma transparente, dos *clusters* sobrecarregados e, em seguida, instancia e/ou encerra *clusters* de acordo com a carga de processamento. Finalmente, o DyLB é responsável

[10] Neste contexto, o termo *cluster* faz referência à infraestrutura física de *hardware* gerenciado por um sistema gerenciador de nuvem computacional.

pelo balanceamento de carga entre os *clusters* disponíveis, considerando o processamento e o estado do fluxo de rede.

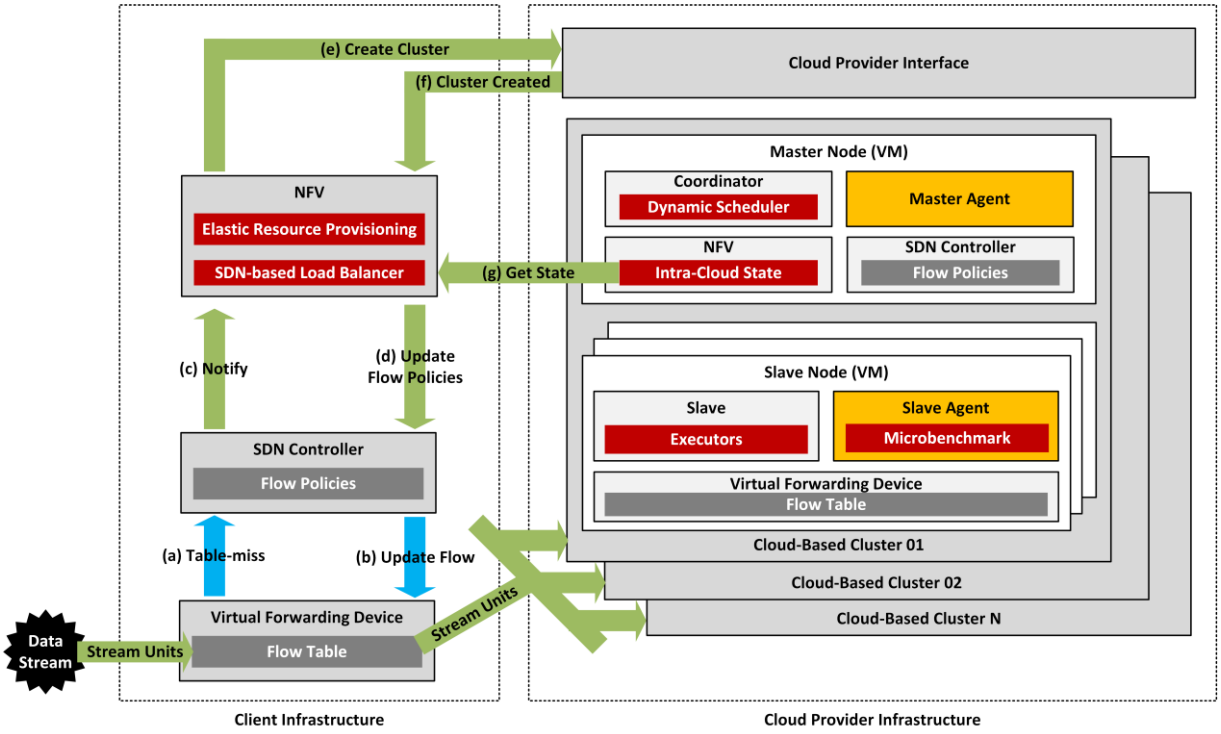


Figura 12 – Visão geral da proposta

A Figura 12, apresenta uma visão geral da proposta, composta pelas soluções: *DySc*, *ERPr* e *DyLB*. As soluções operam de maneira integrada, permitindo assim que toda demanda (*data stream)* seja redirecionada para o destino com maior recurso computacional disponível para processamento. Mais detalhes serão fornecidos nas subseções seguintes.

4.2. Dynamic Scheduler (DySc)

A Figura 13 apresenta o *Dynamic Scheduler* (DySc), que é formado por quatro componentes principais que operam no escalonamento e reescalonamento de tarefas em ambiente *multi-tenant*. Hospedados no *Master Node*, os módulos *Dynamic Scheduler* (DySc) e *Master Agent* (MA) são responsáveis pelo monitoramento do estado computacional dos nós escravos e, baseados nessas informações, realizam o escalonamento e o reescalonamento das tarefas em tempo real (representadas por β_x na Figura 13). Por outro lado, o módulo *Slave Agent*

(SA), hospedado no *Slave Node*, através de *microbenchmarks*, obtém o estado computacional do nó escravo e, quando solicitado, informa o estado ao *Master Agent* (MA). Finalmente, o módulo *Intra-Cloud State*, monitora e fornece um controle fino sobre os fluxos de rede da nuvem em questão. As subseções a seguir detalham cada um dos elementos DySc.

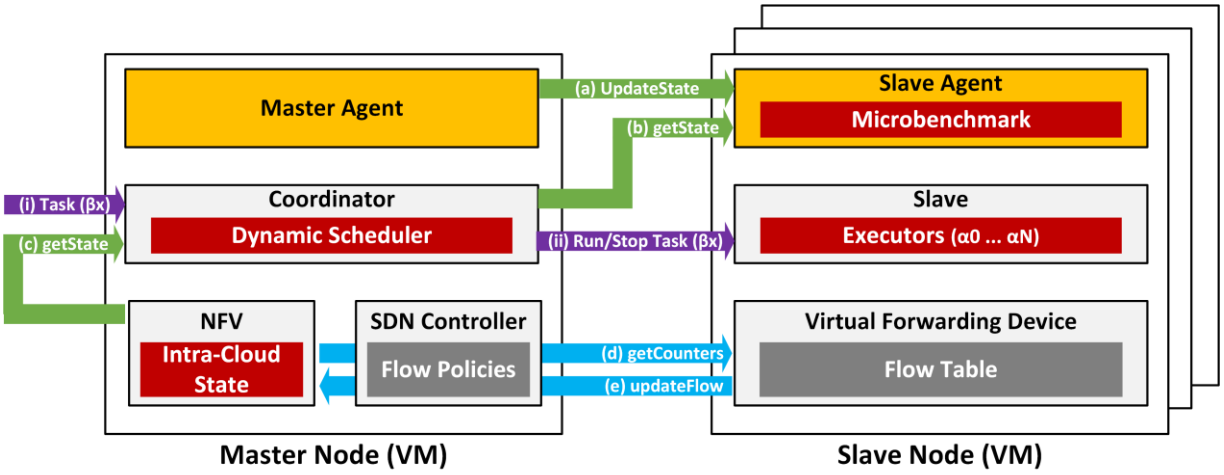


Figura 13 – Visão Geral do *Dynamic Scheduler* (DySc)

4.2.1. Slave Agent (SA)

A proposta considera um ambiente *multi-tenant* com recursos físicos compartilhados. Sendo assim, o *Slave Agent* (SA) é responsável por fornecer o estado de cada nó ($node_{state}$), coletando informações sobre recursos ($resource_{state}$) do sistema. Para cada recurso em questão, dois tipos de estados são coletados pelo SA: o (i) estado físico e (ii) estado virtual. O estado físico se refere à condição atual do recurso compartilhado, como, por exemplo, a utilização do disco. O estado virtual define a condição atual de um recurso interno a VM, como, por exemplo, o consumo de CPU fornecido pelo Sistema Operacional (SO).

Para coletar as informações referentes ao estado físico de cada nó, este trabalho propõe a utilização de *microbenchmarks*. O objetivo é tornar a solução independente de hipervisor e mitigar os conflitos de interesse entre o provedor de serviço da nuvem e o cliente (locatário), uma vez que o provedor pode informar um estado incorreto (e.g.: informar que o cliente sempre possui capacidade máxima de poder de processamento), mesmo quando o cliente encontra-se sobre variações de desempenho [Bouchenak, S., et al, 2013].

O processo de coleta do estado físico dos nós é dividido em duas etapas: estabelecimento da capacidade máxima e avaliação da capacidade. Portanto, na primeira etapa, é estabelecido a capacidade máxima ($resource_{physical\_max}$) de vazão/processamento de cada recurso físico. Essa etapa estabelece como um determinado recurso se comporta sem interferência de entidades externas a máquina virtual. O valor para $resource_{physical\_max}$ pode ser estabelecido através de valores fornecidos pelo fabricante do recurso ou através de ferramentas de *benchmark* em ambientes controlados (sem interferência). Posteriormente, na segunda etapa, através de *microbenchmarks* torna-se possível determinar em tempo real o estado atual aproximado do recurso físico de cada nó ($resource_{physical\_current}$). Dessa maneira, através das variáveis $resource_{physical\_max}$ e $resource_{physical\_current}$, torna-se possível estabelecer o estado aproximado atual de cada recurso físico compartilhado ($resource_{physical\_state}$), conforme denota a fórmula 1.

$$resource_{physical\_state} = \frac{resource_{physical\_current}}{resource_{physical\_max}} \quad (1)$$

É importante destacar que o $resource_{physical\_state}$ pode sofrer alterações devido ao uso interno da própria máquina. Por exemplo, o $resource_{physical\_current}$ de disco pode apresentar uma taxa de escrita/leitura baixa devido à própria utilização interna da VM. Portanto, para considerar tal interferência, utiliza-se o estado interno de um recurso ($resource_{virtual\_state}$) como variável de suavização. O $resource_{virtual\_state}$ é obtido através do Sistema Operacional da VM e define a taxa de utilização de um recurso específico. Por fim, define-se o estado de um recurso ($resource_{state}$) através da fórmula 2.

$$resource_{state} = resource_{physical\_state} + resource_{virtual\_state} \quad (2)$$

Finalmente, o estado de um nó ($node_{state}$) é estabelecido pelo produtório de cada estado dos recursos ($resource_{state}$), definido pela formula 3, em que *n* define a quantidade de recursos.

$$node_{state} = \prod_{i=1}^{n} resource_{state}^{i} \quad (3)$$

4.2.2. Master Agent (MA)

O *Master Agent* (MA) é responsável pelo gerenciamento dos *Slave Agents* (SAs) e da demanda computacional. Para tanto, o MA solicita a atualização dos estados dos nós (Figura 13, *evento a*). No momento que um nó recebe a requisição de atualização de seu estado, o mesmo calcula o seu estado e armazena o valor calculado ($\mathrm{node}_{\mathrm{state}}$). Adicionalmente, o MA requisita de forma periódica o estado de cada nó do *cluster* em questão. Dessa maneira, durante o processo de escalonamento ou reescalonamento (seção 4.2.3-a e seção 4.2.3-b), o MA apenas requisita o estado do nó já computado previamente (Figura 13, *evento b*).

4.2.3. Dynamic Scheduler (DySc)

O *Dynamic Scheduler* (DySc) é responsável pelo escalonamento e reescalonamento de tarefas em um *cluster*. As subseções seguintes descrevem as políticas de escalonamento e reescalonamento de tarefas.

a) Política de Escalonamento (*Scheduling Policy*)

A Tabela 9 exibe o conjunto de notações utilizadas nos algoritmos de escalonamento e reescalonamento. O algoritmo considera que uma tarefa possui um conjunto de unidades de processamento necessárias (β) que precisam ser distribuídas nas unidades de processamento (α) disponibilizadas por um nó *Slave*.

Notação	Descrição
C	*Cluster*
node	Nó do *cluster*
task	Tarefa
β	Unidade de processamento demandada por uma *task*
α	Unidade de processamento de um *Slave*
spreadability	Variável de espalhamento de unidades de processamento da *task*
meaningfulness	Variável que impede reescalonamento quando a diferença de recursos disponíveis não é significativa.

Tabela 9 – Notações utilizadas pelo *Dynamic Scheduler*

O Algoritmo 1 detalha a política de escalonamento do DySc para um conjunto de tarefas (β). O DySc solicita aos SAs a computação do $node_{state}$ para todos os *slave nodes* que compõem o *cluster* (linhas 5 a 7). Os *Slaves* que não possuem α disponíveis são descartados durante o escalonamento (linhas 8 a 12). O DySc então calcula o peso de cada unidade de processamento (weigth_{β}) da tarefa através da soma de todos os $node_{state}$ dividido pelo número de *tasks* (linha 16). O weigth_{β} define o peso de cada unidade de processamento β em relação ao $node_{state}$. Posteriormente, o DySc atribui o β em questão no α do *Slave node* com maior $node_{state}$ ($\text{best}_{\text{node}}$) (linhas 18 e 19). Finalmente, o $best\_node_{state}$ é subtraído de acordo com o weigth_{β} em razão de um fator de espalhabilidade parametrizado (*spreadability*) (linha 20).

Algorithm 1 Scheduling Policy

```
1:  procedure doSchedule(C, task, spreadability)
2:      weight_β ← 0                          ▷ Weight for each assigned β
3:      cluster_score ← 0                     ▷ Sum of all node_state
4:      best_node ← 0                         ▷ Best node in cluster
5:      for all node ∈ C do
6:          node_state ← computeState(node)
7:      end for
8:      for all node ∈ C do
9:          if node_α = 0 then
10:             removeNode(C, node)
11:         end if
12:     end for
13:     for all node ∈ C do
14:         cluster_score ← cluster_score + node_state
15:     end for
16:     weight_β ← cluster_score/task_β
17:     for all β ∈ task do
18:         best_node ← max_node_state(C)
19:         assign_task(β,best_node)
20:         best_node_state = best_node_state - (weight_β * spreadability)
21:     end for
22: end procedure
```

Algoritmo 1 – Política de escalonamento DyS

O valor de *spreadability* define o grau de distribuição dos β entre os *slaves nodes* do *cluster*. Desta maneira, quanto menor o valor de *spreadability*, mais sobrecarregado ficam os melhores *slaves*. Sendo assim, a variável de *spreadability* permite não sobrecarregar o nó com o melhor $node_{state}$ com todos os β. O processo se repete até que todos os β da *task* sejam alocados em um α. O algoritmo de escalonamento de tarefas proposto assume que cada β

demanda recursos dos *slave nodes*, logo evita-se que os melhores *nodes* sejam sobrecarregados através dessa variável.

b) Política de Reescalonamento (*Rescheduling Policy*)

O DySc também é responsável pelo monitoramento de seu *cluster* e pela redistribuição em tempo real dos β previamente escalonados. Essa etapa é denominada reescalonamento. O reescalonamento ocorre de forma periódica, após a etapa de escalonamento. Esse procedimento visa a otimização do desempenho das tarefas, movendo os β dos nós que apresentam baixo valor do $node_{state}$ para os nós que possuem os melhores valores. Essa realocação de tarefas objetiva minimizar a interferência *multi-tenant*. Tal interferência pode ocorrer após o escalonamento de uma tarefa, onde, por exemplo, uma VM de outro inquilino é instanciada no mesmo *hardware* e onera os recursos físicos compartilhados.

A política de reescalonamento que é exibida no Algoritmo 2, assume que as tarefas já estão em execução no *cluster*, ou seja, os β já estão atribuídos a um *slave node*. Dado esse cenário, o DySc solicita aos SAs a atualização dos $node_{state}$ (linhas 7 a 9). Com base nos estados já atualizados, o DySc recalcula o weigth_{β} da tarefa em execução (linha 13). Posteriormente, os $node_{state}$ são atualizados conforme a quantidade de β já alocados nos *nodes* (linhas 14 a 16). O DySc então seleciona dois nós: o *best_node*, que é o *node* com o melhor $node_{state}$ com unidades de processamento (α) disponíveis e o *worst_node*, que é o nó que possui o pior $node_{state}$ com um β alocado (linhas 18 e 19). Caso o $node_{state}$ do *best_node* seja melhor que o $node_{state}$ do *worst_node*, de acordo com um fator de *meaningfulness*, o β é realocado, posteriormente os estados de ambos os nós são atualizados (linhas 20 a 27).

O fator de *meaningfulness* evita possíveis reescalonamentos desnecessários, em que a diferença de recursos disponíveis entre os nós não é suficientemente significativa, e o processo de reescalonamento se repete até que não haja *nodes* disponíveis para realocar unidades β.

Algorithm 2 Rescheduling Policy

```
1: procedure doReschedule(C, task, spreadability, meaningfulness)
2:     weight_β ← 0                          ▷ Weight for each assigned β
3:     best_node ← 0                         ▷ Best node in cluster
4:     worst_node ← 0                        ▷ Worst node in cluster
5:     cluster_score ← 0                     ▷ Sum of all node_state
6:     rescheduling ← true                   ▷ Reschedule verifier
7:     for all node ∈ C do
8:         node_state ← computeState(node)
9:     end for
10:    for all node ∈ C do
11:        cluster_score ← cluster_score + node_state
12:    end for
13:    weight_β ← cluster_score/task_β
14:    for all node ∈ C do
15:        node_state ← updateState(node, weigth_b)
16:    end for
17:    while rescheduling do
18:        best_node = max_node_state_with_avaible_α (C)
19:        worst_node = min_node_state_with_β(C)
20:        if ((best_node_state * meaningfulness) > worst_node_state)
21:          β = getFirstβ(worst_node)
22:          deassign_task(β, worst_node)
23:          assign_task(β, best_node)
24:          best_node_state = best_node_state - (weigth_b * spreadability)
25:          worst_node_state = worst_node_state + (weigth_b * spreadability)
26:        else
27:              rescheduling = false
28:    end while
29: end procedure
```

Algoritmo 2 – Política de Reescalonamento – DySc

4.2.4. Intra-Cloud State NFV

O módulo *Intra-Cloud State NFV* opera no gerenciamento da infraestrutura de nuvem do cliente. Isso permite que o referido módulo NFV seja responsável pela configuração de fluxos de rede intra-nuvem (domínio da infraestrutura do cliente) e também pela definição dos fluxos durante a inicialização (Figura 13, evento e) da infraestrutura através do controlador SDN. Adicionalmente, o *Intra-Cloud State NFV* atua como provedor de estado intra-nuvem, o que permite que os módulos *Dynamic Load Balancer* (Seção 4.3.2) and *Elastic Resource Provisioning* (Seção 4.3.1), determinem o estado da infraestrutura da nuvem do cliente. Para isso, o módulo *Intra-Cloud State NFV* solicita de forma periódica ao DySc a atualização dos

estados dos nós escravos (Figura 13, evento c) e, através do controlador SDN, das métricas de fluxo entre os nós escravos (Figura 13, evento d).

4.3. Provisionamento de Recursos e Balanceamento de Carga

A proposta utiliza a tecnologia NFV através de duas soluções: o *Dynamic Load Balancer* (DyLB) e o *Elastic Resource Provisioning* (ERPr). Os NFV's têm como objetivo proporcionar de forma transparente o balanceamento de carga e provisionamento de recursos para a infraestrutura de processamento (*framework* de processamento de *stream* para *Big Data*) baseada em nuvem, a fim de aprimorar o desempenho das demandas. A proposta considera que um determinado *cluster* que aloca VMs para o processamento de tarefas e, em algum momento, pode tornar-se incapaz de processar uma determinada carga. Isso pode ocorrer devido a um aumento de carga, o que implica em picos de processamento devidos a uma perda significativa no poder de processamento do *cluster* ou provocados pela interferência *multi-tenant*, por exemplo.

Os NFV's dependem do DySc para identificar *clusters* com recursos sobrecarregados (evento g, Figura 12). Para tanto, os NFV's requisitam de forma periódica ao DySc o estado de cada *cluster* ($cluster_{state}$). A fórmula 4 define o estado de um *cluster*, através da média de cada $node_{state}$ dentro do *cluster*. O N denota o número de *nodes* em um cluster.

$$cluster_{state} = \frac{\sum_{N}^{i} node_{state}^{i}}{N} \quad (4)$$

Baseado no $cluster_{state}$ de cada *cluster*, os NFV's provêm elasticidade de recursos e o balanceamento de carga. As subseções a seguir descrevem de forma detalhada o funcionamento do provisionamento elástico de recursos (ERPr) e a solução para balanceamento de carga (DyLB).

4.3.1. Elastic Resource Provisioning (ERPr)

A solução adotada para o provimento de elasticidade é em escala horizontal a nível de *cluster*, em que o *cluster* é replicado, diferentemente da abordagem horizontal (tradicional) onde são inseridos novos *slave nodes (slave nodes,* Figura 13*)* para o *cluster*. Dessa forma,

pretende-se fornecer recursos de forma elástica sem conhecimento por parte da aplicação. Dada essa propriedade, o ERPr se baseia nos valores de $cluster_{state}$ a fim de identificar o estado da infraestrutura ($infrastructure_{state}$). A equação 5 define o estado de uma infraestrutura que é representada pela média do estado de cada $cluster_{state}$ dentro de uma infraestrutura. O N denota o número de *clusters* em uma infraestrutura.

$$infrastructure_{state} = \frac{\sum_{N}^{i} cluster_{state}^{i}}{N} \quad (5)$$

O ERPr instancia ou desativa um *cluster* de acordo com a $infrastructure_{state}$. Para tanto, o ERPr utiliza dois valores parametrizados: o $infrastructure_{lower\ threshold}$ e $infrastructure_{upper\ threshold}$. O valor de $infrastructure_{lower\ threshold}$ define o estado mínimo aceitável de $infrastructure_{state}$. Nesse caso, quando o $infrastructure_{state}$ é inferior ao $infrastructure_{lower\ threshold}$, o ERPr requisita para a plataforma de nuvem a criação de um novo *cluster* (evento e, Figura 12). Em contrapartida, quando $infrastructure_{state}$ é maior que $infrastructure_{upper\ threshold}$, o ERPr requisita para a plataforma de nuvem o encerramento do *cluster* que possui o pior $cluster_{state}$ (evento e, Figura 12).

O provisionamento elástico de recursos é executado de forma periódica de acordo com a soma de dois parâmetros: T e $cluster_{creation\ time}$. O T é um valor parametrizado que define a frequência das atualizações do estado do *cluster*, enquanto o $cluster_{creation\ time}$ é o tempo exigido para que a plataforma de nuvem instancie o *cluster*. Ao utilizar ambos os parâmetros, a função de provisionamento elástica de recursos aguarda o momento de criação do *cluster* (evento f, Figura 12), antes de verificar o $infrastructure_{state}$ novamente.

É importante destacar que por se tratar de carga de processamento de *data stream* para *Big Data*, esse trabalho assume que a carga de processamento permanecerá em operação por um período de tempo superior ao tempo gasto para provisionar novos recursos ($cluster_{creation\ time}$).

4.3.2. Dynamic Load Balancing (DyLB)

O provisionamento elástico de recursos (ERPr) expande a capacidade de processamento de uma infraestrutura através da criação de múltiplos *clusters* para o processamento da carga. Entretanto, a unidade geradora de fluxo (*data stream*, Figura 12) de dados (evento e, Figura 12) não é capaz de estabelecer para qual *cluster* a demanda será encaminhada. Com intuito de realizar esse encaminhamento de forma transparente através de um balanceador de carga, o presente trabalho depende do modelo SDN.

A solução de balanceamento de carga estabelece a capacidade de carga do cluster ($cluster_{load\ capacity}$) de acordo com o $cluster_{state}$. Cada $cluster_{load\ capacity}$ é estabelecido de forma periódica conforme a fórmula 6, em que N denota o número de *clusters* em uma infraestrutura.

$$cluster_{load\ capacity} = \frac{cluster_{state}}{\sum_{N}^{i} cluster_{state}^{i}} \quad (6)$$

O método de balanceamento de carga desempenha sua função baseando-se na granularidade de unidades de *stream* (*stream unit*, Figura 12). Cada *cluster* recebe durante um determinado período de tempo o valor percentual de carga de acordo com $cluster_{load\ capacity}$. Por exemplo, um *cluster* formado por ferramentas de processamento para *stream* (e.g. *Apache Storm*), com capacidade $cluster_{load\ capacity} = 0.7$, deve receber 70% das unidades de *stream* (e.g. Tuplas *Storm*). O valor de $cluster_{load\ capacity}$ é atualizado de acordo com uma janela pré-determinada de tempo.

A Figura 12 ilustra o método de balanceamento de carga. O fluxo de dados *stream* gera periodicamente unidades de *stream* e envia ao *cluster* conhecido. Como na primeira requisição não existe entrada de fluxo especificada para as unidades de *stream* (*stream unit*), de acordo com seus identificadores (e.g. IP de origem e porta), ocorre o chamado *table miss* (ausência de regra para um determinado fluxo). Dessa forma, o dispositivo de encaminhamento virtual (SDN *Switch*) encaminha esse primeiro fluxo para o Controlador SDN (*evento a*, Figura 12). Em seguida, o Controlador SDN notifica o DyLB (*evento c*, Figura 12) para estabelecer qual *cluster* deve processar a unidade de *stream* de acordo com $cluster_{load\ capacity}$ de cada *cluster*. A partir disso, o DyLB atualiza as políticas de fluxo (*evento d*, Figura 12). Em seguida, o Controlador

SDN cria uma entrada de fluxo para o *cluster* selecionado para receber a carga de unidade de *stream* (*evento b*, Figura 12).

4.4. Discussão do Capítulo

Esse capítulo dissertou sobre o conjunto de soluções que constituem a proposta da presente tese de doutorado. A seção 4.2 apresentou o DySc, que é um escalonador e reescalonador de tarefas baseado em *microbenchmarks* para avaliar, em tempo real, o estado computacional dos nós que compõem o *cluster*. Isso possibilita que a demanda seja encaminhada para nós com maior poder computacional disponível no momento do escalonamento e durante a execução da tarefa, através do reescalonamento. A solução ERPr (subseção 4.3.1) tem o propósito de iniciar e/ou encerrar um novo conjunto de máquinas virtuais (duplicar o *cluster* existente) sob um novo conjunto de máquinas físicas, para isso é necessário interagir diretamente com o gerenciador de nuvem. O emprego do ERPr se faz essencial, uma vez que o conjunto de máquinas virtuais que realizam o processamento das tarefas podem vir a experimentar o compartilhamento de *hardware* (*multi-tenant*). Consequentemente, pode implicar na queda de desempenho por parte da aplicação. Dessa maneira, a partir do instante que mais de um *cluster* se encontra nessa situação, faz-se necessário direcionar as tarefas entre esses ambientes. Como a aplicação não realiza essa distribuição, existe a necessidade de aplicar um balanceador de carga, que nesse trabalho foi intitulado DyLB (sub-seção 4.3.2). O DyLB realiza a distribuição de tarefas entre os *clusters* existentes, com o objetivo de distribuir tarefas considerando o poder computacional total do *cluster*. Para isso, a solução direciona as tarefas priorizando os *clusters* com maior poder computacional.

O conjunto de soluções apresentado nesse capítulo pode ser empregado em diferentes tecnologias, precisando apenas de ajustes de implementação para cada cenário. Sendo assim, a proposta não é dependente das tecnologias utilizadas na avaliação (Capítulo 5).

Capítulo 5

Protótipo e Avaliação

Este capítulo tem como propósito descrever a aplicação e avaliação do modelo proposto, que neste momento será chamado de protótipo do trabalho. O protótipo é divido em (i) ERPr (*Elastic Resource Provisioning*), (ii) DyLB (*Dynamic Load Balancing*) e (iii) DySc (*Dynamic Scheduler*).

As soluções propostas neste trabalho (ERPr, DyLB e DySc) podem ser aplicadas em qualquer plataforma de nuvem e de *Big Data*, o que dependerá de ajustes de implementação para operação em cada plataforma.

5.1. Estudo de Caso Preliminar

Para dar suporte a motivação principal do trabalho, que é a detecção de interferência *multi-tenant*, inicialmente será apresentado um estudo de caso preliminar que avalia esse impacto sobre o *framework* de *Big Data Apache Storm*.

5.1.1. Cenário

Uma série de testes foram conduzidos para diagnosticar problemas existentes que motivam a solução proposta. Para esse propósito, um ambiente controlado de computação em nuvem foi construído utilizando o *framework HPE Eucalyptus* (versão 4.3.0). O ambiente de computação em nuvem é composto por 4 computadores físicos, cada um equipado com CPU Intel i7 de 8 núcleos, 16GB de RAM, e conectados através de interface de rede *gigabit*. O *host* é equipado com o S.O. CentOS 7.0, sendo que as VMs executam o S.O. Ubuntu Server 16.04 através do *hypervisor* KVM (*Kernel-based Virtual Machine*). A nuvem computacional foi dividida em duas zonas, cada uma composta por duas máquinas físicas. A infraestrutura da

nuvem do cliente foi constituída por cinco maquinas virtuais (um *master* e quatro *slave nodes*), que foram atribuídas as máquinas físicas de acordo com a política de nuvem padrão do Eucalyptus (*Round Robin*).

O *framework Apache Storm*, versão 1.0.2, foi utilizado na infraestrutura em nuvem do cliente. Foi desenvolvida uma ferramenta que realiza a leitura das informações relacionadas a geração e processamento das tuplas de fluxo de dados. O *Storm UI REST* fornece essa informação [Storm UI REST, 2014] o que permitiu a interação com o *cluster Storm.* Que por sua vez, possibilitou o gerenciamento de suas operações, como iniciar ou encerrar uma topologia, e fornecer informações sobre o estado atual das topologias em execução. A ferramenta desenvolvida requisita o número de tuplas processadas a cada 10 segundos, que é a periodicidade de atualização padrão do *Storm* para esses valores.

Para as avaliações, foram utilizadas três topologias, de forma que cada topologia utilizasse um recurso computacional específico. Para geração de carga de CPU, a topologia *Word Count* (WC) [Word Count Topology, 2013] foi utilizada. Para avaliar o impacto de uso de rede, utilizou-se a topologia *Throughput Test* (TT) [Storm Throughput Test, 2012], que gera cargas com conteúdo aleatório de tamanho fixo de 10 KB. Finalmente para avaliar o impacto no disco, realizou-se a mesma abordagem de [Xu, J., et al, 2014], onde a topologia WC foi customizada para que as tuplas realizassem a leitura das palavras via disco e a escrita destas palavras também ocorressem no disco. Neste trabalho, essa topologia customizada é chamada de *Word Count File* (WCF).

No que diz respeito às topologias e seus parâmetros, foram realizadas avaliações de comportamento em dois cenários distintos, que é apresentado pela Figura 14. De maneira sucinta, o cenário foi divido em *(i) Baseline Cluster* e *(ii) Multi-Tenant Cluster*.

- *Baseline Cluster:* uma máquina física hospeda um *Master Node* e um *Worker Node*, os demais nós hospedam os *Worker Nodes*, que compõem o *Cluster* principal.
- *Multi-Tenant Cluster*: esse cenário tem como objetivo avaliar o comportamento das topologias ao mesmo tempo que experimentam experiência *multi-tenant*. Similar ao *Baseline Cluster*, cada máquina física hospeda as VMs que formam o *cluster* principal. No entanto, duas máquinas físicas hospedam um *cluster Storm* secundário (Figura 14, *multi-tenant*

cluster). Assim esse cenário é composto por dois *clusters*: o *cluster* principal, composto por cinco VMs (distribuídas em todas as zonas), e o *cluster* secundário composto por cinco VMs (distribuídas somente na zona B do *Eucalyptus*). Esse cenário permite que uma avaliação do desempenho das topologias do *cluster* principal sob interferência *multi-tenant* em 50% das máquinas físicas.

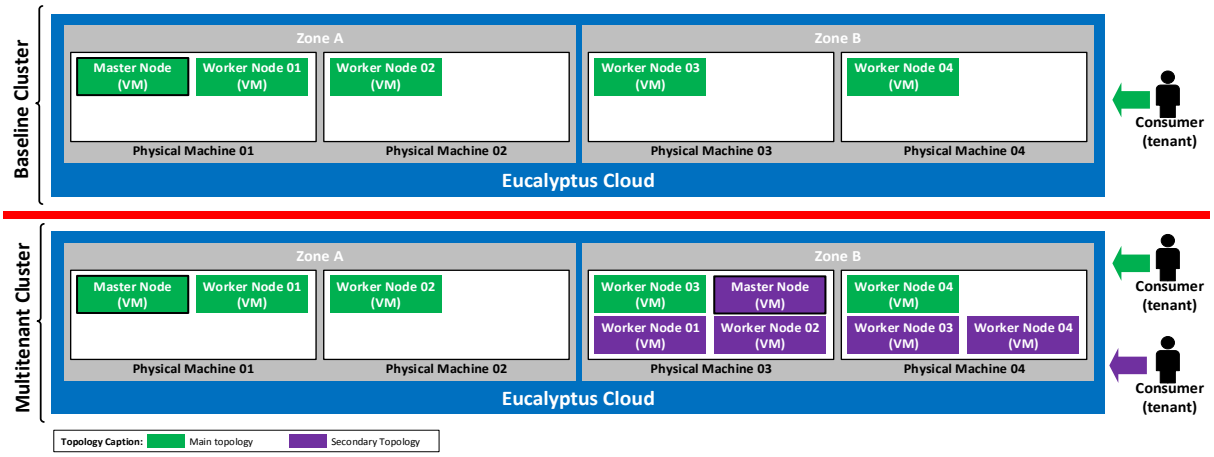


Figura 14 - Cenário de Avaliação

A Figura 14 apresenta a distribuição do *Baseline Cluster* e *Multi-Tenant Cluster*. A avaliação das métricas de desempenho foram medidas em relação ao *cluster* principal. O *cluster* secundário é responsável pela geração paralela de carga de processamento, que ocorre através da execução de topologias mantendo os mesmos parâmetros e configurações do *cluster* principal. Dessa forma, a configuração adotada imita o ambiente do mundo real, no qual uma máquina física pode hospedar diversas máquinas virtuais com diferentes prioridades e variados serviços.

5.1.2. Avaliação de Desempenho

Essa subseção apresenta os resultados da avaliação do impacto *multi-tenant* sobre a estrutura de nuvem em tempo real. O tempo de execução para cada teste foi de 300 segundos. Os parâmetros das topologias são apresentados na Tabela 10.

Tabela 10 - Configuração das topologias para cluster principal e secundário.

Topologies	***Number of Workers***	***Spout (Sentence)***	***Bolt (Split)***	***Bolt (Count)***
Baseline Cluster	4	4	8	12
Multi-tenant Cluster	4	4	8	12

Os resultados são apresentados na Tabela 11, em que a quantidade média de tuplas processadas por segundo em ambos os cenários são fornecidas para cada topologia avaliada. O *Baseline Cluster* representa o número de tuplas geradas por segundo em um ambiente livre de interferência *multi-tenant*. A coluna *Multi-Tenant Cluster* lista a porcentagem de tuplas geradas por segundo quando o ambiente experimenta a interferência *multi-tenant* comparada ao *Baseline Cluster*. A avaliação permite notar que o desempenho das topologias é deteriorado em todos os casos quando em experiência *multi-tenant.* Porém, de forma substancial quando ocorreu execução da mesma topologia em paralelo (espelhamento).

Tabela 11 - Comparação entre cenários principal e secundário.

Topologias	**Baseline Cluster (tuplas/s)**	**Multi-tenant Cluster (Porcentagem relacionada ao Baseline)**		
		WC (CPU)	**WCF (Disco)**	**TT (Rede)**
WC (CPU)	803,622	**42.7%**	35.8%	82.5%
WCF (Disco)	5,757	80.9%	**33.5%**	89.4%
TT (Rede)	1,333	80.4%	36.9%	**53.1%**

Em todos os casos que as topologias utilizaram o mesmo tipo de recurso computacional, o desempenho foi significativamente prejudicado. Na topologia WC (consumo de CPU), apenas 42.7% das tuplas foram processadas quando executadas sobre interferência *multi-tenant*. Para a topologia WCF (consumo de disco), apenas 33,5% das tuplas foram processadas. Finalmente, para a topologia TT (consumo de rede), apenas 53,1% das tuplas foram processadas quando comparadas aos resultados do *Baseline Cluster*.

Os resultados (Tabela 11) revelam a importância de métodos de escalonamento que consideram o estado dos nós físicos a fim de evitar impactos no desempenho em aplicações de *Big Data* em Nuvem. Os ambientes *multi-tenant* são cenários realistas, especialmente na computação em nuvem (e.g., AWS, Salesforce, etc.), que influenciam drasticamente no desempenho do processamento de *data stream* para *big data*, como mostraram as avaliações sobre o *Apache Storm*. Essa situação demanda cautela, uma vez que tipicamente os provedores

de infraestrutura em nuvem vendem a infraestrutura como serviço (*Infrastructure as a Service - IaaS*), e não fornecem qualquer informação sobre o estado do *hardware* físico que executa a VM a nível de IaaS.

Assim, durante o processo de distribuição das tarefas, o algoritmo de escalonamento deve considerar o estado físico do nó a fim de evitar a degradação do desempenho da aplicação. Adicionalmente, o processamento de *data stream* em *frameworks* de *big data*, também devem considerar o reescalonamento periódico de tarefas, uma vez que o desempenho das VMs pode mudar drasticamente ao longo do tempo, em decorrência do acesso simultâneo ao *hardware* por parte de inquilinos terceiros da nuvem.

5.2. Protótipo ERPr and DyLB

A Figura 15 ilustra a aplicação das soluções ERPr e DyLB. A fim de tornar a avaliação do protótipo realista, foi implementado um gerador de fluxo de dados (Figura 13, *Data Stream*), cuja função é gerar, de maneira periódica, tuplas para uma topologia. Essa metodologia possibilita que a carga seja gerada fora da topologia, como ocorre em aplicações de produção no mundo real. Floodlight (Versão 1.2) [Floodlight, 2012] foi utilizado no protótipo como controlador SDN e os nós foram equipados com o OpenVSwitch (versão 2.8.0). Um módulo PACKET-IN foi implementado no controlador, sendo que a função desse módulo é a receber as mensagens PACKET-IN (que ocorre quando um pacote não possui políticas de fluxo correspondentes) do OpenVSwitch a cada nova chegada de tupla (Figura 15, *Stream Units*), identificadas através de endereços IP e porta de origem. O módulo solicita o DyLB e aguarda uma resposta em relação a qual cluster o pedido de geração de tupla deve ser encaminhado.

HPE Helion Eucalyptus (versão 4.3.0) foi utilizado como provedor de nuvem. Cada ambiente de computação em nuvem é composto por quatro computadores físicos equipados com CPU Intel i7 de 8 núcleos e 16GB de RAM, conectados através de uma interface de comunicação de rede Gigabit. A infraestrutura em nuvem do cliente (Figura 12, *Cloud-Based Cluster*) foi criada através de um *template CloudFormation* [HPE, 2014]. O *template* cria cinco máquinas virtuais (um *master* e quatro *workers*), e esses nós executam dinamicamente a topologia do *Apache Storm* através da ferramenta *cloud-init*.

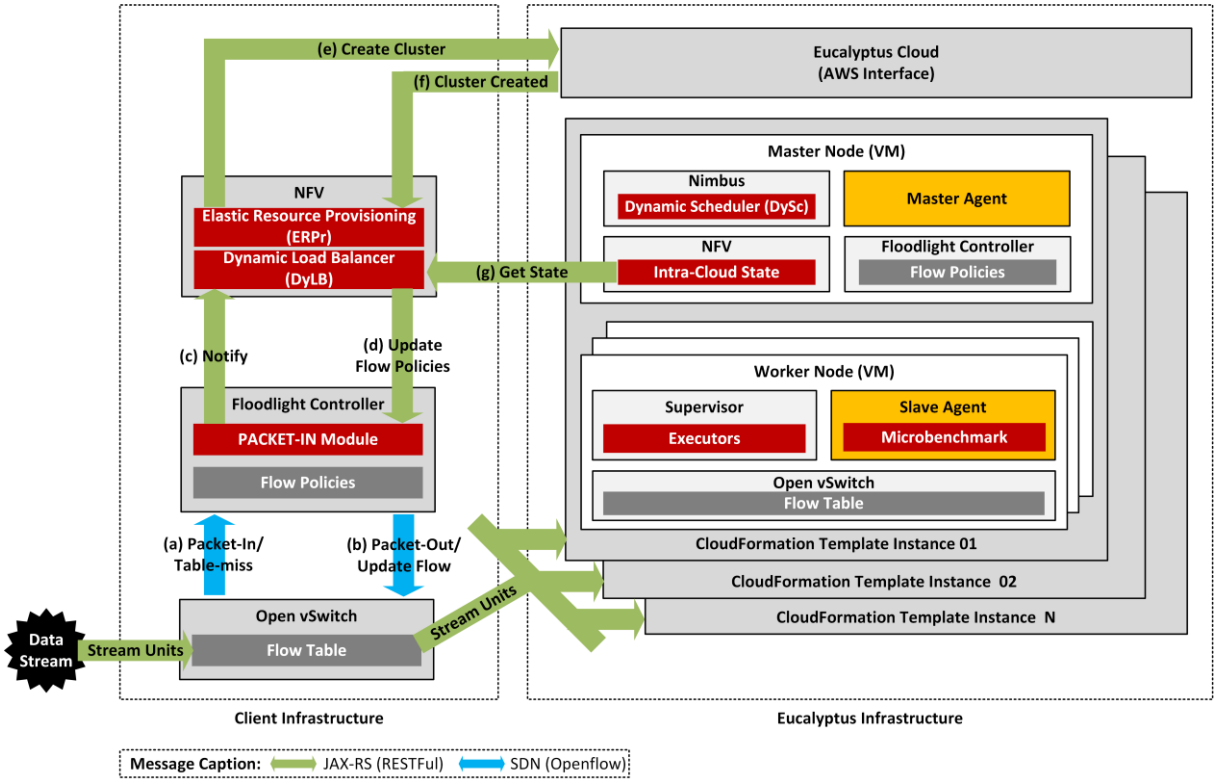


Figura 15 - Protótipo de implementação ERPr e DyLB

O módulo DyLB solicita periodicamente o estado computacional ao *Intra-Cloud State NFV*, para cada instância do *CloudFormation* com intervalos de 60 segundos. O módulo ERPr igualmente solicita ao DyLB que atualize o $cluster_{state}$ a cada 60 segundos. O DyLB recebe as notificações PACKET-IN do controlador SDN, que são chamadas através do módulo PACKET-IN implementado no *Floodlight Controller*. O módulo de balanceamento de carga DyLB é definido no nível de tuplas (nível de aplicação) de acordo com os endereços IP e porta de origem. O ERPr requisita ao controlador de nuvem (*Eucalyptus Cloud*) a criação ou término de *clusters* através da API cliente do *Eucalyptus* (Figura 15, evento e/f). A configuração do *cluster* é construída a partir de um *Eucalyptus CloudFormation template*.

5.3. Protótipo DySc

Essa seção descreve o protótipo DySc executado no *template CloudFormation*. É importante relembrar que toda a infraestrutura é dinamicamente configurada através do campo *cloud-init* do *CloudFormation template*, possibilitando a avaliação do ERPr. A Figura 16

apresenta a arquitetura do protótipo DySc. Uma instância virtual do *OpenVSwitch* é executada em cada *worker*, o que permite a avaliação fina (avaliado na subseção 5.6.1) de fluxos de rede internos na nuvem em questão. A configuração do fluxo é executada na inicialização do *template CloudFormation*, através da Interface REST do controlador *Floodlight*. Os contadores de fluxo também são obtidos através da interface REST do controlador *Floodlight*. Os contadores são invocados a cada 60 segundos.

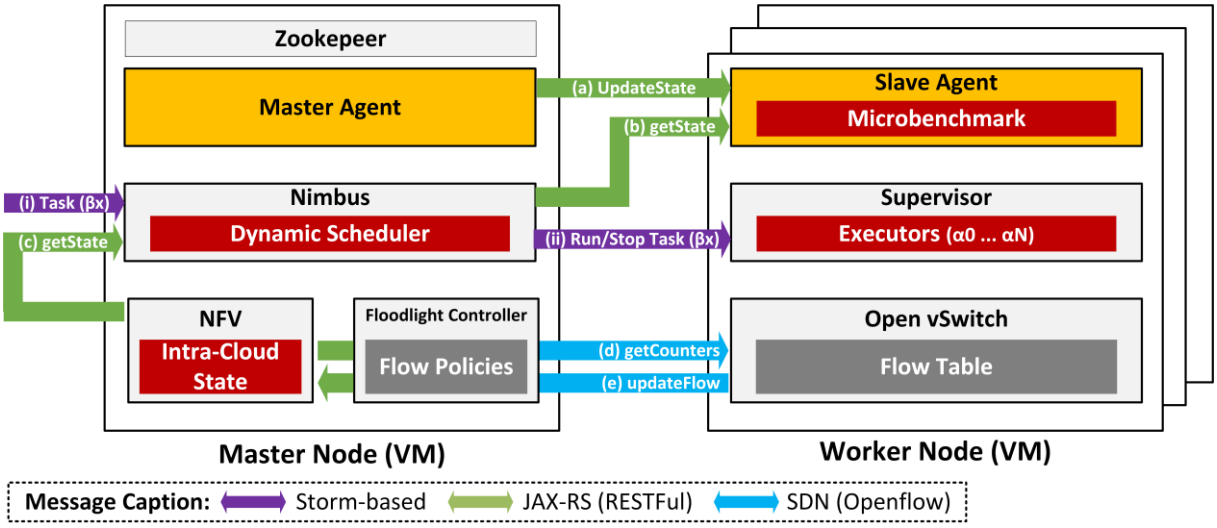


Figura 16 - Arquitetura de implementação do protótipo DySc.

Conforme mencionado na seção 2.1.2, o *framework Apache Storm* permite que a política de escalonamento seja personalizada, possibilitando o desenvolvimento de novas políticas de escalonamento conforme as necessidades do usuário/ambiente. A API que fornece essa customização é chamada de *IScheduler*, que fornece um método de escalonamento que possibilita o desenvolvimento customizado da política de escalonamento.

O MA (*Master Agent*) é executado no nó *Master Node*, da topologia *Storm*. Já nos *worker nodes*, são executados os SAs (*Slave Agents*) que foram implementados como um *web service* RESTful através da JAX-RS API [JAX-RS, 2007]. Dessa forma, os SAs possuem dois métodos principais: (i) *updateState* (Figura 16, evento a), responsável pelo início do cálculo do estado computacional do nó, e (ii) *getState* (Figura 16, evento b), que é responsável por retornar o último estado computacional calculado. O DySc inicia ou interrompe uma tarefa (um conjunto de unidades de processamentos β) através das funções *assign* e *freeSlot* do *Apache Storm* [Storm, 2016]. As notações que foram definidas e utilizadas na arquitetura DySc e Storm (Figura 16) podem ser consultadas na Tabela 9.

Para calcular o estado computacional dos nós em termos de CPU, Disco e Rede, foram utilizados *microbenchmarks*. Os recursos computacionais em questão foram compartilhados em um ambiente *multi-tenant* e, portanto, sofreram degradação da capacidade de processamento/ vazão quando sobrecarregados por outro cliente na nuvem, conforme já discutido na seção 5.1. A Tabela 12, apresenta os métodos e ferramentas utilizados para obter o estado dos recursos compartilhados (resource $_{\text{physical\_state}}$).

Tabela 12 – Métodos computacionais para obtenção do estado dos recursos

Recurso	**Processo para obter o $resource_{physical\_state}$**
CPU	Sysbench [Sysbench, 2009] executa um conjunto de operações para computar números primos. O tempo médio de processamento com os recursos livres foi previamente estabelecido ($resource_{physical\_max}$) e o tempo real ($resource_{physical\_state}$) foi calculado como segue: $$resource_{physical\_state} = 1 - \frac{resource_{physical\_current} - resource_{physical\_max}}{resource_{physical\_max}}$$
Disco	Sysbench [Sysbench, 2009] executa leituras e gravações de um conjunto predeterminado de blocos. O tempo médio de escrita com recursos livres foi estabelecido (resource$_{physical_max}$) e o tempo real ($resource_{physical\_state}$) foi calculado como segue: $$resource_{physical\_state} = \frac{resource_{physical\_current}}{resource_{physical\_max}}$$
Rede	O *Master Node* hospeda o iPerf [iPerf, 2003] em modo servidor. Assim mede-se o *throughput* máximo para cada *Slave Node*, que por sua vez possui o cliente iPerf. A taxa média de *throughput* foi previamente estabelecida ($resource_{physical\_max}$) e o tempo real ($resource_{physical\_state}$) foi calculado como segue: $$resource_{physical\_state} = \frac{resource_{physical\_current}}{resource_{physical\_max}}$$

De forma a garantir que as medições dos recursos computacionais físicos sejam executados em tempo real, o MA conta com um algoritmo cíclico para atualizar o $node_{\ state}$ (Figura 16, evento a). O MA aguarda por um tempo predeterminado antes de solicitar a próxima atualização do nó, impedindo que os *worker nodes* realizem *microbenchmarks* simultâneos, evitando que ocorra a degradação do estado físico ou tenham resultados influenciados pelo armazenamento em cache de dados.

O $resource_{virtual\_state}$ de CPU, disco e rede são obtidos através do Sistema Operacional Linux utilizando o sistema de arquivos *proc* [PROC, 1994].

As ferramentas utilizadas permitem a parametrização dos tempos de execução do *microbenchmark*. Tal preocupação se justifica porque o objetivo não é medir o tempo total de execução para a computação dos recursos físicos utilizando *benchmarks* de longa duração. O que é desejado é a realização de *microbenchmarks* para obter resultados em tempo real e utilizar adequadamente os recursos disponíveis em um ambiente *multi-tenant*.

5.4. Avaliação

Esta seção relata as avaliações das soluções propostas: DySc, DyLB e ERPr.

5.4.1. Avaliação Dynamic Scheduler - DySc

O desempenho do DySc foi avaliado e comparado com *EvenScheduler* (ES), escalonador padrão utilizado pelo *Apache Storm*, que opera com a política de escalonamento *round-robin*. O objetivo foi avaliar o DySc em cenário com sobrecarga (multi-tenant). Os experimentos foram realizados no cenário de testes (Figura 14).

As mesmas topologias utilizadas para avaliar o *EvenScheduler* (Seção 5.1) foram utilizadas para avaliar o DySc. Conforme apresentado na Tabela 11, todas as topologias avaliadas (WC, WCF e TT) sofreram interferência *multi-tenant*, e o ES não identificou a interferência em nenhum caso. O que se espera é que o algoritmo de escalonamento proposto (DySc) melhore o desempenho dessas topologias.

O DySc necessita de dois parâmetros, conforme discutido na Seção 5.3. Para o parâmetro *spreadability* foi utilizado o valor 1.0 para aumentar a distribuição das tarefas entre os nós com melhores estados computacionais. Para o parâmetro *meaningfulness*, foi utilizado o valor de 0.5, estabelecido através da análise de testes do comportamento do algoritmo de reescalonamento. O tempo para reescalonamento executado pelo DySc é de 60 segundos, o que permite que a atualização dos estados dos SAs. Adicionalmente, é fornecido 15 segundos para que cada SA atualize seu estado, após esse período, o MA solicita que o próximo nó atualize seu estado, e assim sucessivamente. No momento que os estados de todos os nós forem atualizados, o MA aguarda mais 15 segundos antes de reiniciar o processo de atualização.

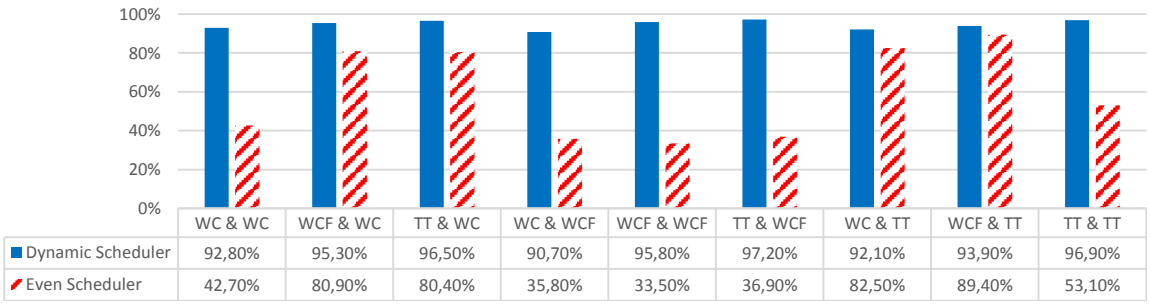
Avaliação do Escalonador

Esta subseção apresenta os resultados obtidos durante a execução das políticas ES e DySc em um ambiente *multi-tenant*. As medidas comparam o número total de tuplas geradas

para o DySc e ES no Cluster Base (*Baseline Cluster* - Figura 14), o qual não há interferência de VM terceiras.

A Figura 17 compara o desempenho das topologias. O eixo vertical representa a porcentagem relativa máxima de tuplas processadas em comparação com o *Baseline Cluster*. O eixo horizontal representa os pares da topologia, a primeira topologia é executada no cluster principal e segunda topologia executada no cluster secundário (*Multi-Tenant Cluster* - Figura 14), gerando cargas de processamento paralelas para imitar a interferência *multi-tenant*.

Quando as topologias são "espelhadas" em clusters (os mesmos recursos estão sobrecarregados), o DySc forneceu a melhoria de 50,10% quando o WC (consumo de CPU) é executado em paralelo tanto no cluster principal quanto no secundário (WC & WC da Figura 17). Nas demais topologias, DySc processou 62,30% mais tuplas que ES na topologia WCF (WCF & WCF da Figura 17 - consumo de Disco) e 43,80% mais tuplas ao executar a topologia TT (TT & TT da Figura 17 - consumo de Rede).



	WC & WC	WCF & WC	TT & WC	WC & WCF	WCF & WCF	TT & WCF	WC & TT	WCF & TT	TT & TT
Dynamic Scheduler	92,80%	95,30%	96,50%	90,70%	95,80%	97,20%	92,10%	93,90%	96,90%
Even Scheduler	42,70%	80,90%	80,40%	35,80%	33,50%	36,90%	82,50%	89,40%	53,10%

Figura 17 - Comparação de tuplas processadas entre DySc e ES.

Quando o recurso principal utilizado pela topologia está disponível, mas outro recurso está com interferência *multi-tenant*, o DySc pode melhorar o desempenho do ES. Por exemplo, na topologia WC (consumo de CPU) quando executado a topologia WCF (consumo de Disco) no cluster secundário. Adicionalmente, é possível analisar que o escalonador ES apresenta um desempenho significativamente menor quando a mesma topologia está em execução em ambos os clusters, visto que ambas as topologias necessitam do mesmo tipo de recurso. Outro fato importante que deve ser destacado é que quando o recurso de disco é utilizado por outros inquilinos da nuvem, o desempenho das topologias reduz de maneira significativa, processando em média 35,40% da tuplas, enquanto o DySc processar uma média de 94,57% em relação ao cluster base.

A Figura 18, ilustra o tempo médio de processamento para cada tupla da topologia WC, quando utilizados os escalonadores DySc e ES. DySc realizou o processamento das tuplas com tempo médio de 0,00014 ms, enquanto ES exigiu aproximadamente 0,00022 ms. Assim, o emprego do DySc no escalonamento provocou uma melhoria de 57% no tempo de processamento por tupla.

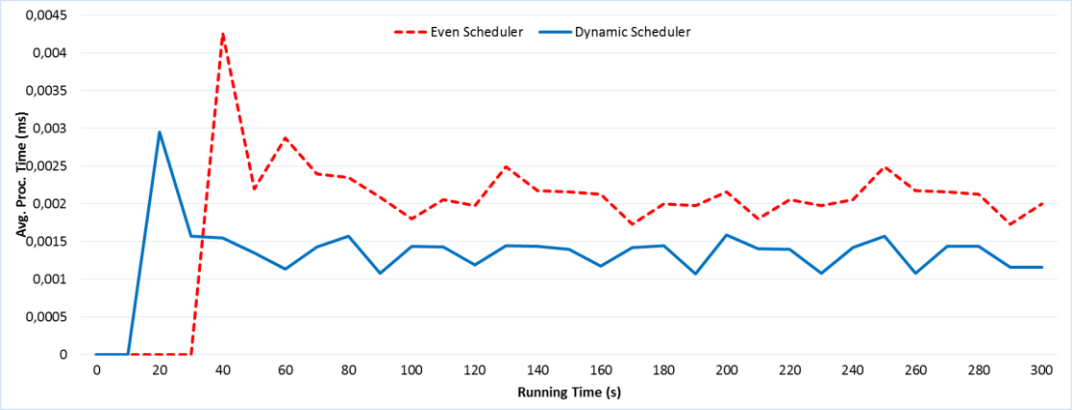


Figura 18 - Topologia WC (CPU), tempo de processamento por tupla.

A Figura 19, apresenta o tempo médio de processamento para cada tupla da topologia WCF. DySc obteve uma melhora de 172% em relação a ES (média de 0,18 ms contra 0,49 ms).

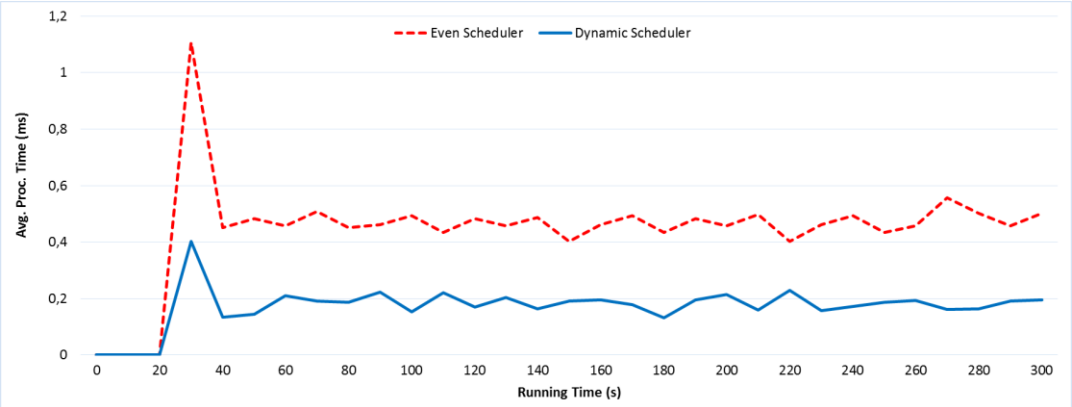


Figura 19 - Topologia WCF (Disco), tempo de processamento por tupla.

A Figura 20 ilustra o tempo médio de processamento para cada tupla com a topologia TT. DySc apresentou novamente melhores resultados quando comparado ao ES. O tempo médio de processamento para tupla foi de 0,78 ms para o DySc, contra 1,52 ms para o ES, que representa uma melhoria de 94,87%.

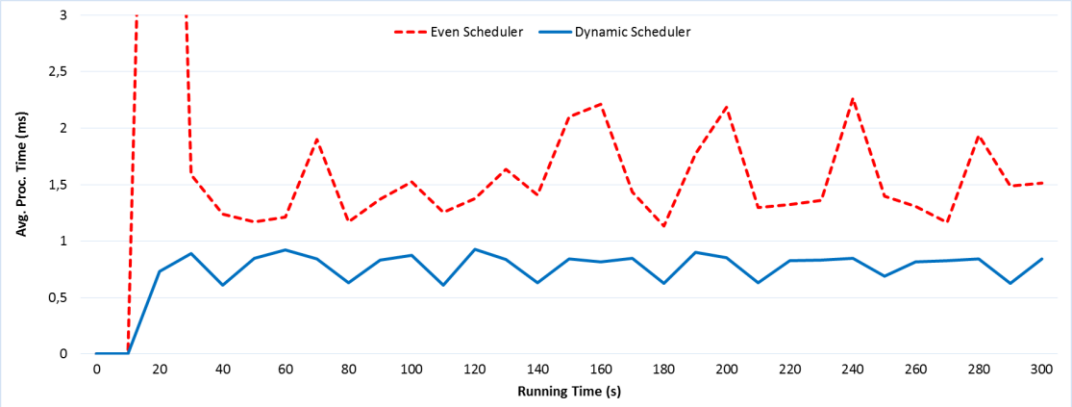


Figura 20 - Topologia TT (Rede): tempo de processamento por tupla.

O comportamento do ES apresenta uma maior variação quando comparado ao DySc. Isso é reflexo da ausência de políticas de escalonamento de avaliação do estado computacional de um nó de maneira prévia ao envio de uma solicitação de processamento. *Frameworks* de *Streaming* para *Big Data* geram uma grande quantidade de dados de maneira intermitente, e é esperado que o ambiente de processamento não ofereça um tempo de processamento altamente variável.

Avaliação do Reescalonador

Ambientes *multi-tenant* são tipicamente caracterizados pelo uso variável de recursos computacionais ao longo do tempo. Portanto, as cargas de processamento foram variadas com o propósito de avaliar o comportamento do Reescalonamento do DySc, em face a mudanças dos estados dos nós. O objetivo é que a abordagem proposta seja capaz de identificar a degradação dos recursos, efetuar a troca da carga de processamento entre os nós do cluster, e redistribuir as tarefas imediatamente após os recursos se encontrarem disponíveis novamente.

O ambiente controlado utilizado na avaliação consiste em um cenário de reescalonamento que contém um cluster sem interferência *multi-tenant* e um cluster com interferência *multi-tenant*, intitulados (i) *Baseline Cluster* e (ii) *Multi-Tenant Cluster*, ambos ilustrados na Figura 14. As execuções foram divididas em três momentos distintos: (i) recursos computacionais totalmente disponíveis (*Baseline Cluster*); (ii) processamento paralelo iniciado

em 50% dos nós físicos, sobrecarregando seus recursos computacionais (Figura 14 - *Multi-Tenant Cluster*, topologia secundária iniciada) e (iii) processamento paralelo encerrado, retornando ao estado de disponibilidade de recursos iniciais (*Baseline Cluster*). Cada período foi executado por 1200 segundos.

Para medir os impactos e benefícios do Reescalonamento DySc, foi realizado um teste utilizando a topologia *Throughput Test* (TT), para processamento com e sem interferência (*Multi-Tenant Cluster* e *Baseline Cluster*). A Figura 21 apresenta o tempo médio de processamento de tuplas de DySc e ES, que executam a topologia TT no cenário de reescalonamento. O processamento paralelo iniciou em 1200 segundos. DySc necessitou de 60 segundos para identificar a alteração de processamento e executar o reescalonamento. Após 2400 segundos, o processamento paralelo finalizou e o DySc necessitou de 90 segundos para identificar a alteração e redistribuir as tarefas para todas as máquinas do *cluster*. É possível otimizar o tempo de reescalonamento forçando os nós a atualizarem seus estados com maior frequência (15 segundos, por exemplo), ou ainda se o processo de reescalonamento for executado com uma frequência menor (a avaliação considerou 60 segundos). O tempo total necessário para o reescalonamento (identificar e redistribuir) em ambas as mudanças de processamento correspondeu a apenas 4% do tempo total da execução.

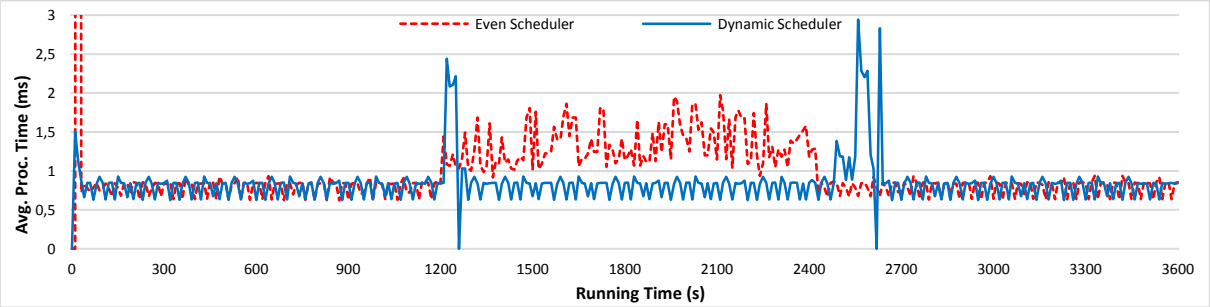


Figura 21 - Avaliação Reescalonamento DySc e ES

5.4.2. Avaliação Dynamic Load Balancing - DyLB

Para avaliar o DyLB, um cenário composto de duas nuvens computacionais foi configurado: o *framework HPE Eucalyptus Cloud* foi utilizado para configurar o referido cenário e cada nuvem computacional utilizou 4 máquinas físicas. Uma nuvem continha todos seus recursos físicos disponíveis (representando assim o *Baseline Cluster*, Figura 14), enquanto

a segunda nuvem continha 50% de seus recursos físicos executando demandas a partir de outro cliente (representando assim o *Multi-Tenant Cluster*, Figura 14).

A topologia de consumo de CPU (WC) foi executada em ambos os clusters, e as requisições de processamento de tuplas foram geradas de maneira intermitente (Figura 15, *Stream Units*) fora da infraestrutura da nuvem. Para cada requisição de processamento de tuplas, ocorre o que a literatura chama de *Table-Miss* (falta de correspondência na tabela de fluxo) e portanto uma notificação ocorre (Figura 12), gerando assim uma requisição ao DyLB para determinar para qual cluster a tupla deve ser encaminhada para processamento. A fórmula 6 foi utilizada para calcular a carga de cada cluster. Foi definido um intervalo de 60 segundos para a atualização dos valores de $cluster_{load\ capacity}$. A distribuição de cargas entre os clusters durante a execução do cenário é ilustrada na Figura 22.

Figura 22 - Avaliação da distribuição de carga entre clusters.

Como avaliação geral, é possível notar que a abordagem de balanceamento de carga proposta distribuiu a carga corretamente, visto que apenas 27,96% da carga de processamento (em média) foi encaminhada ao cluster que sofria interferência *multi-tenant* (*Multi-Tenant Cluster*, Figura 14). Já o cluster que não sofreu interferência multi-tenant (*Baseline Cluster*, Figura 14), e consequentemente possuía recursos disponíveis, recebeu 72,04% (em média) da carga de processamento durante a avaliação. Dessa forma, é possível concluir que a proposta DyLB é em média 22,04% mais eficaz que a abordagem tradicional *Round-Robin*, que realiza uma distribuição igualitária dentre os clusters disponíveis.

5.4.3. Avaliação Elastic Resource Provisioning - ERPr

Finalmente, a proposta inteira foi avaliada, composta por DySc, DyLB e ERPr. Para tanto, duas nuvens *Eucalyptus* foram implantadas, cada nuvem composta por quatro máquinas físicas (Figura 14). O cenário de teste foi executado por 9000 segundos. Nos primeiros 3000 segundos a primeira nuvem não experimentou interferência *multi-tenant*, porém, após 3000 segundos, a primeira nuvem começou a sofrer interferência (topologia WC concorrente executada em todas as máquinas físicas), e esse período de interferência também durou 3000 segundos. Finalmente, nos últimos 3000 segundos (6000 a 9000 segundos de execução do cenário) a interferência *multi-tenant* causada pela primeira nuvem *Eucalyptus* encerrou e o período final de teste ficou livre de interferência.

Um único cluster foi implantado na primeira nuvem *Eucalyptus*, que executou a topologia WC, enquanto a segunda nuvem *Eucalyptus* permaneceu disponível para o ERPr alocar novos *clusters* através de um *template CloudFormation*. Os parâmetros utilizados foram: $infrastructure_{lower\ threshold}$ definido com 0.3 e $infrastructure_{upper\ threshold}$ definido com 0.9, uma vez que proporcionou a criação e encerramento de *cluster* de maneira satisfatória através de testes de avaliação. A Figura 23 ilustra os resultados da avaliação realizada para a distribuição de carga realizada pelo DyLB, o tempo médio de processamento do DyLB (para todos os clusters alocados) e o atraso do ERPr para identificação do início da *interferência multi-tenant* e da finalização.

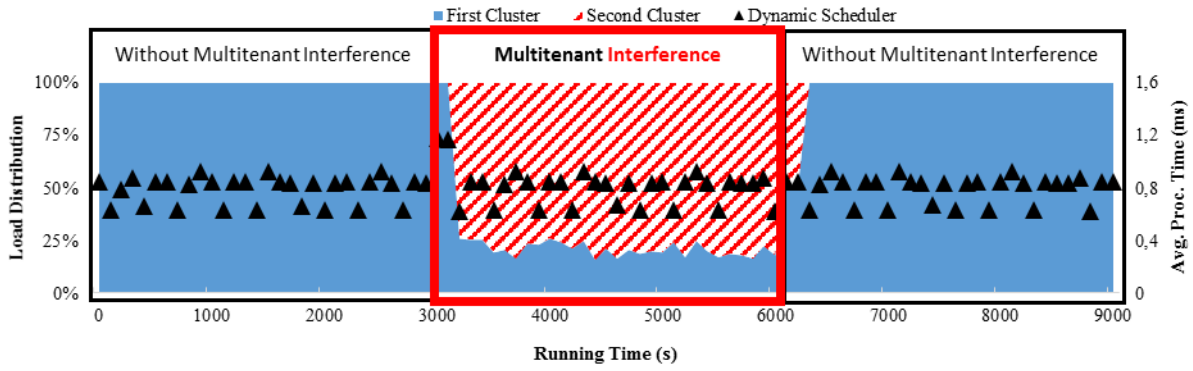


Figura 23 – Avaliação ERPr com interferência *multi-tenant* destacada (borda vermelha).

O tempo médio de processamento por tupla durante o teste não apresentou variações significativas. Em 3000 segundos, momento em que o primeiro cluster sofre interferência *multi-*

tenant, o ERPr identificou estado de interferência através da variável $infrastructure_{state}$, e na sequência requisitou a criação de um novo *cluster* (segunda nuvem *Eucalyptus*). Foi observado um atraso de 107 segundos até a criação do segundo cluster, que é o tempo demandado pela segunda nuvem *Eucalyptus* para instanciar o *template CloudFormation*. A partir do momento que o segundo cluster foi criado, a solução DyLB proporcionou um equilíbrio na distribuição de carga de maneira que o segundo cluster recebeu em média 78% de toda carga gerada. Finalmente, em 6000 segundos, a interferência terminou e o ERPr exigiu 225 segundos para encerrar o segundo cluster. Para isso, foi utilizado os valores da variável $infrastructure_{upper\ threshold}$.

5.5. Discussão Prévia

A interferência ocasionada pelo compartilhamento de recursos em ambientes *multi-tenant* impacta significativamente na capacidade de processamento das infraestruturas computacionais que operam nesse contexto. A fim de mensurar esse impacto, uma infraestrutura de nuvem computacional privada, semelhante a um ambiente de nuvem pública foi construída (Figura 14). As avaliações demonstraram que o compartilhamento de recursos impacta em até 57%, 66% e 46% para aplicações dependentes de CPU, disco e rede respectivamente. Tais resultados motivaram notoriamente a proposta apresentada na presente tese.

Com propósito de avaliar a proposta no contexto de compartilhamento de *hardware* (*multi-tenant*), inicialmente foi avaliada a solução responsável pelo o escalonamento e reescalonamento de tarefas, intitulada *Dynamic Scheduler*. As avaliações demonstraram que o escalonador proposto é capaz de identificar máquinas virtuais com recursos físicos sobrecarregados, de maneira independente de hipervisor e sem acesso a métricas de consumo físico do *hardware* (Figura 17). Adicionalmente, a abordagem de reescalonamento foi capaz de identificar mudanças do estado físico da máquina virtual ao longo do tempo (Figura 21). Logo, as abordagens de escalonamento e reescalonamento propostas identificaram de maneira satisfatória máquinas virtuais em situações de *multi-tenant*.

Posteriormente, a abordagem proposta de balanceamento de carga, *Dynamic Load Balancing*, foi avaliada. Para isso, duas nuvens computacionais foram executadas, sendo que

uma nuvem encontrava-se sob experiência *multi-tenant*, enquanto a segunda encontrava-se em estado normal (livre de interferência). O balanceador de carga proposto, através das métricas obtidas pelo *Dynamic Scheduler*, foi capaz de identificar corretamente o *cluster* sobrecarregado, e distribuir a carga corretamente (Figura 22). Finalmente, a proposta foi avaliada integralmente através do *Elastic Resource Provisioning*. As avaliações demonstraram a viabilidade da proposta, que possibilitou (Figura 23): identificar *cluster* sobrecarregado, instanciar um *cluster* adicional, o balanceamento adequado de carga entre os *clusters* e, finalmente, identificar a possibilidade de encerramento do *cluster* adicional.

Após testar a viabilidade da proposta, identificou-se a possibilidade de aprimorar alguns aspectos, mais especificamente na identificação do estado *multi-tenant*, uma vez que a proposta baseia-se na utilização de *microbenchmarks* para identificar o referido estado. Esse procedimento utiliza ferramentas que provocam consumo desnecessário de recursos computacionais de um *cluster* computacional, uma vez que, para cada identificação de possíveis problemas, as ferramentas de *microbenchmark* devem ser executadas novamente.

Considerando a limitação citada, a seção seguinte apresenta alternativas para identificar o compartilhamento de *hardware* nesse cenário de nuvem computacional.

5.6. Alternativas ao Micro-benchmark

Conforme já discutido na seção 3.4, é possível identificar a variação de desempenho através de ferramentas de *benchmark*, porém isso gera um custo computacional adicional e implica em duas principais desvantagens: (i) a degradação de desempenho causada pelo processamento dos *microbenchmarks*; e (ii) desperdício de recursos para execução dos *microbenchmarks*. Diante desses fatos, buscou-se alternativas para mitigar esses impactos no momento da avaliação dos recursos computacionais.

A primeira proposta de solução (subseção 5.6.1) apresentada como alternativa ao *microbenchmark* utiliza os contadores de fluxos que a tecnologia SDN oferece. Na sequência, é apresentada a segunda solução (subseção 5.6.2) que aplica aprendizagem de máquina no momento da avaliação dos recursos computacionais.

5.6.1. Avaliação de Fluxo SDN

Com o objetivo de minimizar os impactos relacionados ao *microbenchmark*, foram considerados os contadores de fluxo SDN dos *Worker Nodes*, no qual é possível medir a largura de banda ponto-a-ponto (*Worker-to-Worker*). A hipótese é que devido à natureza homogênea de processamento distribuído dos *frameworks* de *big data*, a taxa de dados entre os nós também deve ser homogênea, pois as cargas de processamento são distribuídas igualitariamente na maioria dos casos. Assim, a avaliação de interferência *multi-tenant* pode ser identificada através da análise do contador de fluxo, eliminando a necessidade de realizar *microbenchmark* durante o processo de escalonamento.

Para avaliar essa hipótese, os contadores de fluxo do controlador *Floodlight* foram mensurados. Foram avaliados os dois cenários: (i) *Baseline Cluster* e *Multi-Tenant Cluster* (Figura 14). A avaliação foi realizada durante 3600 segundos em ambos os cenários. A Figura 24 e Figura 25 ilustram as taxas de *download* entre os *Worker Nodes*, enquanto a topologia TT (*Throughput Test*) é executada em ambos os *clusters (Baseline e Multi-Tenant)*.

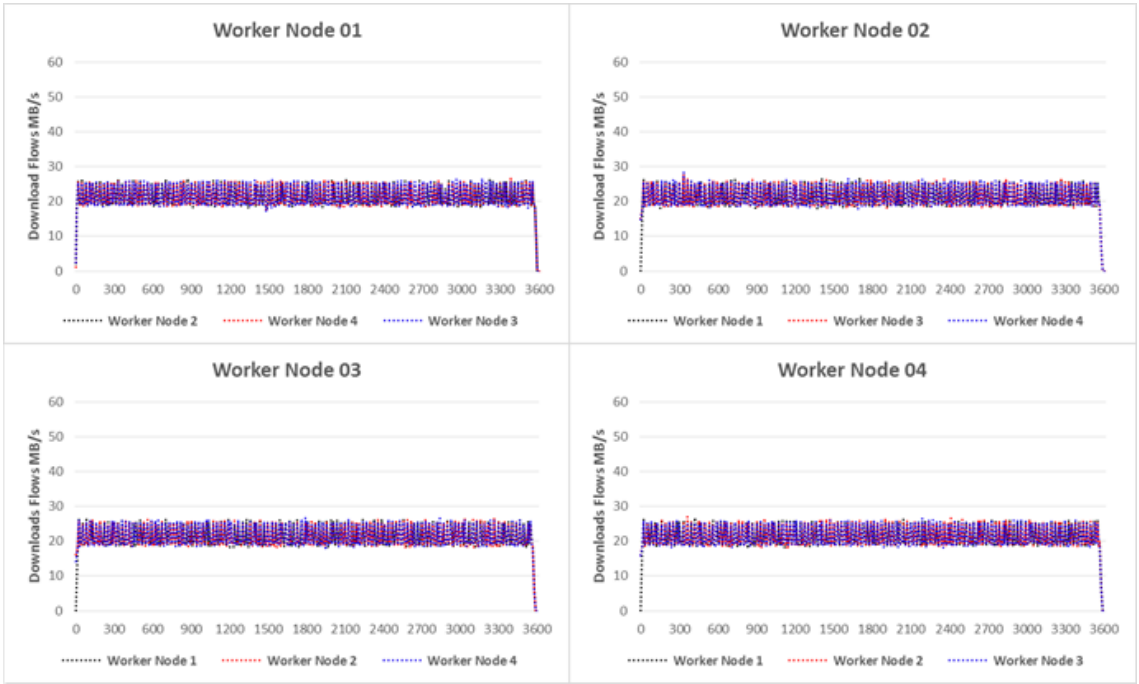


Figura 24 - Taxas de *Download* entre *Worker Nodes* (*Baseline Cluster*)

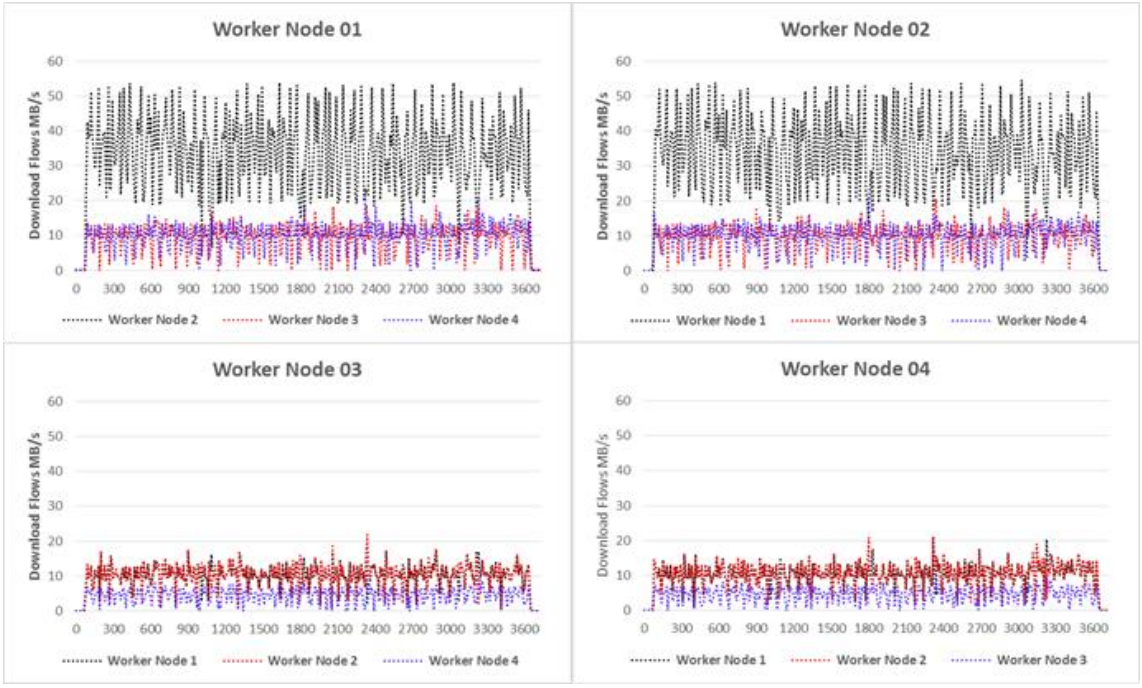


Figura 25 - Taxas de *Download* entre *Worker Nodes* (*Multi-Tenant Cluster*)

É possível observar que os fluxos de rede entre os *Worker Nodes* variam significativamente quando executados no cenário *multi-tenant*. A taxas de *download* entre os *Worker Nodes* 01 e 02 são substancialmente superiores quando comparadas com as taxas medidas no cluster base (*Baseline Cluster*). Isso aponta que os nós estavam sobrecarregados no cenário *multi-tenant e* que os recursos dos *Worker Nodes* 03 e 04 foram degradados. Ou seja, as taxas de *download* entre os *Worker Nodes* 03 e 04 foram consideravelmente inferiores quando comparadas com os *Worker Nodes* 01 e 02. Finalmente, no cenário *Baseline Cluster* as taxas de *download* entre os *Worker Nodes* não apresentaram variações significativas.

A Figura 26 apresenta as taxas médias de *download* entre os *Worker Nodes* em ambos os cenários de teste (*Baseline e Multi-Tenant Cluster*). As linhas azuis apresentam as taxas de *download* entre os *Worker Nodes* 01 e 02 no cenário m*ulti-tenant*, com média de 35,72 MB/s e 35,43 MB/s, respectivamente. No cenário sem interferência, as taxas médias de *download* entre os *Worker Nodes* permanecem semelhantes, apresentando uma diferença de no máximo 0,6 MB/s.

Esses resultados mostram que ao analisar os fluxos de rede entre os *Worker Nodes*, via controlador SDN, torna-se viável a identificação de VMs que estão sob interferência *multi-tenant.* Adicionalmente, tal prática pode eliminar a necessidade da aplicação de

microbenchmarks para identificar interferência *multi-tenant*, reduzindo desperdício de recursos e degradação de desempenho.

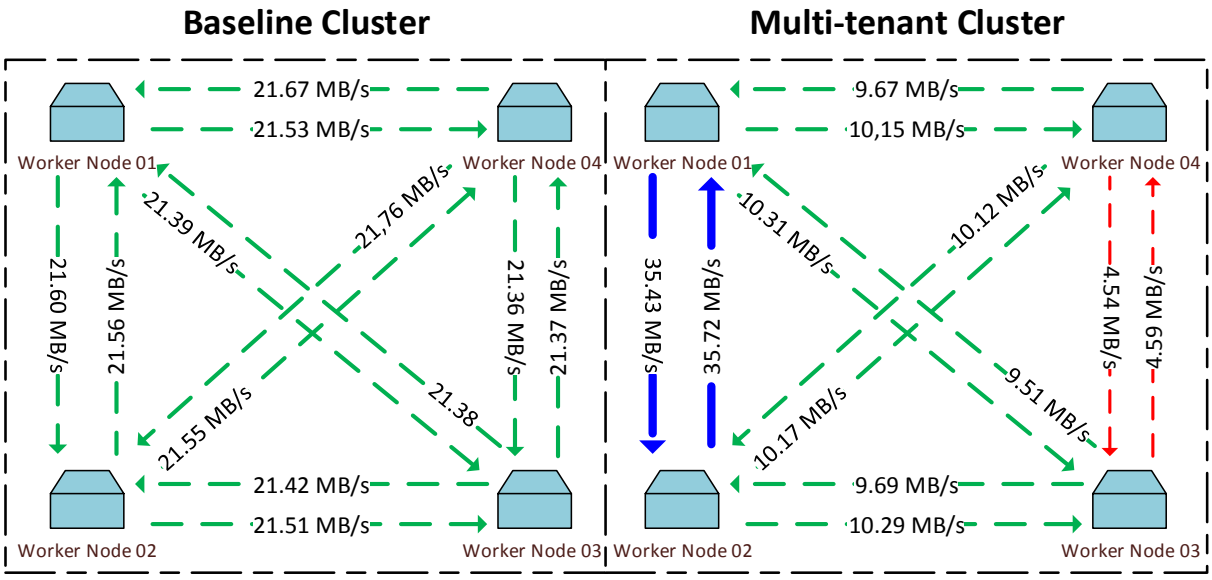
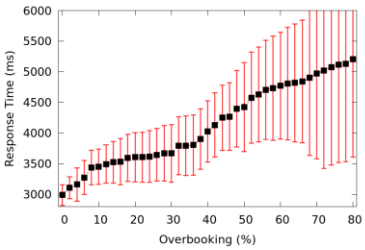
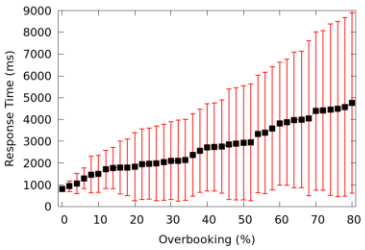


Figura 26 - Taxas de *download* entre *Worker Nodes* (VMs) nos cenários *Baseline e Multi-Tenant Cluster*

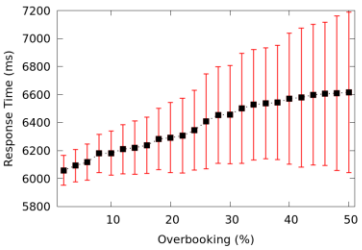
5.6.2. Identificação de Multi-Tenant via Aprendizagem de Máquina



(a) Topologia CPU-bound; nível de overbooking é medido através do steal-time de CPU.



(b) Topologia Disk-bound; nível de overbooking é medido através do wait time de CPU.



(c) Topologia Network-bound; nível de overbooking é medido a nível de hypervisor, através do monitoramento da utilização de rede do tenant

Figura 27 - Impacto de interferência *multi-tenant* na aplicação *Apache Storm.*

A segunda alternativa proposta para identificar interferência *multi-tenant* é aplicação de Aprendizagem de Máquina no contexto da pesquisa. Para isso foi proposto um modelo de auditoria de dois níveis para identificar as interferências no domínio do inquilino, que conta

com técnicas de aprendizagem de máquina alimentada através de métricas oriundas da aplicação e recursos virtuais.

A fim de identificar a relação entre o tempo de resposta médio de cada *tupla* e grau de *overbooking*, realizou-se uma avaliação preliminar (Figura 27) utilizando as topologias que consomem CPU, Disco e Rede. Para a aplicação de CPU, o tempo médio de resposta é de apenas 2989ms quando o fator de *overbooking* é zero. No entanto, a medida que o que o grau de *overbooking* aumenta, o tempo de resposta também aumenta (de 74% a 80% é o aumento no tempo de resposta).

O mesmo comportamento é evidenciado nas demais topologias, onde o tempo de resposta aumenta em média 482% para disco e 9% para rede. Independentemente do tempo de médio de resposta das topologias, o desvio padrão aumenta significativamente.

A) Arquitetura

A Figura 28 apresenta a arquitetura da proposta de monitoramento, onde um cliente da nuvem pretende monitorar sua aplicação distribuída (e.g. *Apache Storm*). Para avaliação, foi considerado o mesmo cenário dos estudos anteriores. Porém, foram desenvolvidos dois monitores para coleta de informações: *(i) Application Monitor* e *(ii) Virtual Resources Monitor*. O *Application Monitor* é responsável pela coleta periódica de métricas de desempenho de cada executor da topologia (Seção 2.1.2), enquanto o *Virtual Resources Monitor* é responsável pela coleta periódica de métricas acerca dos recursos virtuais no domínio do inquilino (e.g. CPU *Steal Time*). Ambos os monitores enviam, de maneira periódica, as métricas coletadas para o *Auditing Agent*, que realiza a auditoria de desempenho dos recursos.

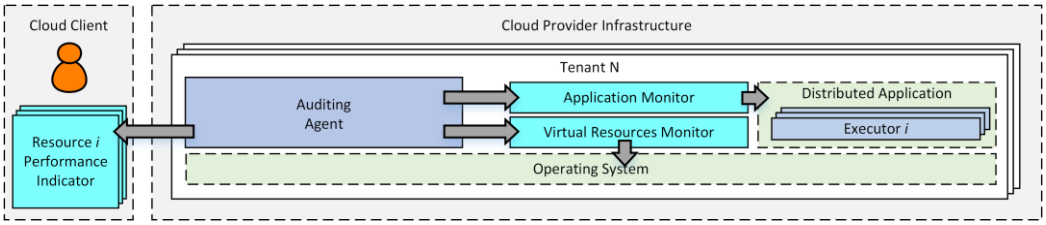


Figura 28 - Arquitetura do modelo de autenticação em dois níveis

O objetivo do *Auditing Agent* (Figura 29) é identificar se as métricas de desempenho coletadas foram obtidas em um ambiente livre de interferência *multi-tenant*, e também identificar a interferência *multi-tenant* de fato.

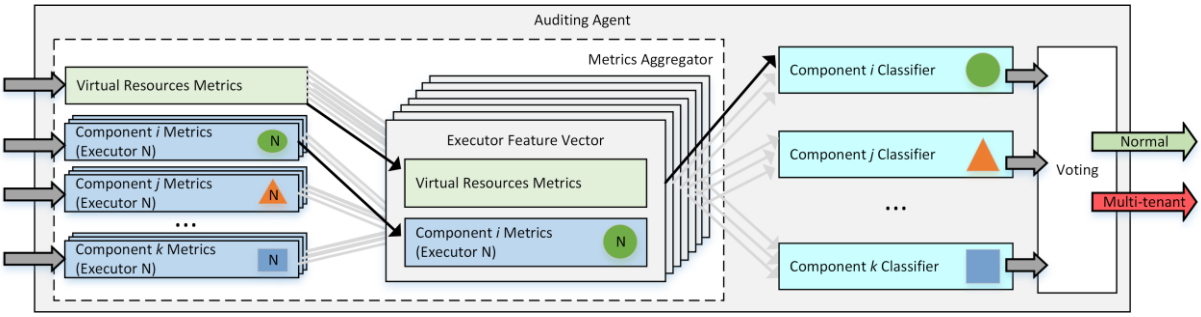


Figura 29 – Modelo do *Auditing Agent*

Para identificar interferência *multi-tenant*, o *Auditing Agent* emprega técnicas de aprendizagem de máquina sobre as métricas coletadas pelo *Application Monitor e Virtual Resources Monitor*. O agente utiliza as métricas de desempenho coletadas pelos monitores como entrada. Para métrica de desempenho de cada *executor*, o agente gera um vetor de características que é composto pelas métricas do executor e pelas métricas virtuais correspondentes.

Depois de gerar o vetor de características, o processo de classificação é executado. Assim, os vetores de características são fornecidos aos classificadores de acordo com o tipo de componente (*spout*, *split* ou *bolt*). Finalmente, a classe *(Normal ou Multi-Tenant)* é atribuída através de um esquema de votação. Dessa forma, os recursos provenientes de um *tenant* é classificado como *Normal* somente se a maioria de seus executores forem classificados como *Normal*, caso contrário, assume-se que existe interferência *multi-tenant*.

B) Protótipo

A Figura 30 apresenta o protótipo da proposta com aprendizagem de máquina. A avaliações consideraram a mesma estrutura de *Big Data* para processamento de *stream*, com o *framework Apache Storm*, utilizada nos testes anteriores. *Apache Storm* é executado em nuvem, onde cada inquilino possui *slots* e cada *slot* possui um conjunto de executores.

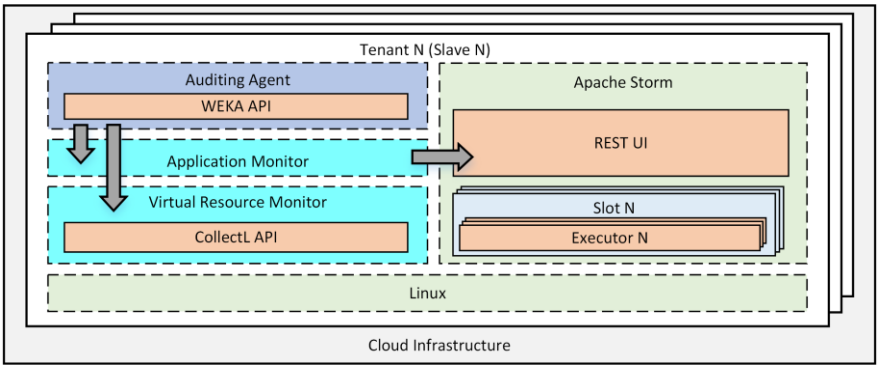


Figura 30 - Protótipo do Modelo

A solução *Application Monitor* coleta periodicamente as métricas de cada executor via REST UI [Storm UI REST, 2014]. Já a solução *Virtual Resources Monitor* coleta as informações através da API CollectL [Collectl, 2005]. Um total de 7 métricas foram coletadas da aplicação (executores da topologia) e 11 métricas acerca dos recursos virtuais (via *CollectL*). A Tabela 13 apresenta as métricas coletadas.

Tabela 13 - Métricas coletadas para avaliação.

Feature Group	Features
Application-based	Number of input tuples; Number of output tuples; Average delay; Average processed tuples per second; Difference from last number of input tuples; Difference from last number of output tuples; Difference from last average delay; Difference from last average processed tuples per second
Virtual-Resources-based	CPU load; Average CPU load last 1 min; Average CPU load last 5 min; KB read from disk; KB written to disk; Disk write requests; Disk read requests; Network packets received; Network packets sent; Network data received; Network data sent;

A solução de aprendizagem de máquina *Auditing Agent* foi implementada utilizando a ferramenta Weka API [Weka, 2006], e as métricas são coletadas a cada 10 segundos e posteriormente é criado um vetor de característica.

C) Avaliação

Para avaliar essa alternativa ao *microbenchmark*, dois cenários em nuvem foram considerados: nuvem privada e nuvem pública. A nuvem privada considerada é o mesmo cenário de avaliação da seção 5.1.1, já para nuvem pública a avaliação ocorreu na *Amazon AWS*

[Amazon AWS, 2006]. Para avaliar os recursos de CPU, disco e rede, as topologias WC, WCF e TT, foram aplicadas.

Os classificadores são treinados na nuvem privada para simular a interferência *multi-tenant*, e a mesma topologia (espelho) é executada em paralelo em outro inquilino no mesmo *host* físico. Para isso, foram criados três conjuntos de treinamento distintos: CPU, disco e rede. Para o processo de classificação, dois classificadores distintos foram utilizados: *Naive Bayes* (NB) e *Decision Tree* (DT). Para NB, foi aplicado o processo de discretização supervisionado e para DT aplicou-se o algoritmo J48.

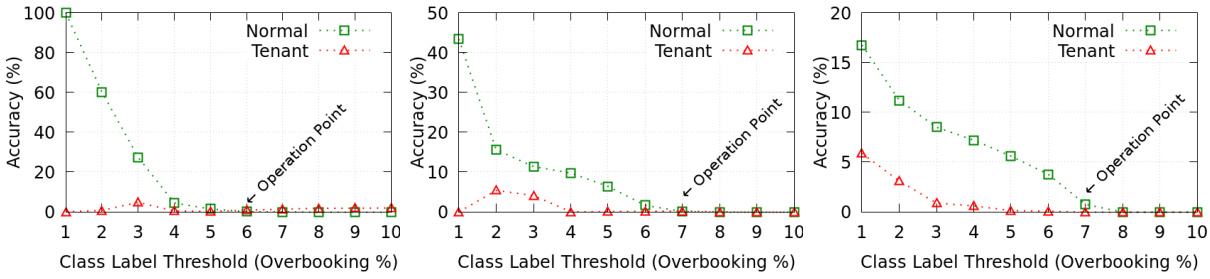
O processo de avaliação objetivou responder às seguintes questões de pesquisa: (i) *Qual é quantidade mínima de interferência multi-tenant necessária de acordo com cada recurso para que o modelo classifique adequadamente o estado dos nós?* (ii) *Quão desafiador é realizar a classificação em diferentes configurações de hardware?* (iii) *Como a solução proposta atua em nuvem pública, quando seu modelo foi construído em um ambiente controlado em nuvem privada?*

Para responder à questão (i), a primeira avaliação teve como objetivo definir a melhor limiar entre as classes *Normal* e *Multi-Tenant*. Para isso, os rótulos foram definidos de acordo com as taxas de *overbooking* de recursos, no qual o inquilino (nó) só é considerado *Normal* se a taxa de *overbooking* for menor que a limiar definida, caso contrário é considerado *Multi-tenant*. A Figura 31 apresenta a relação de taxas de Falso Positivo (taxas de instâncias *Normais*, que foram classificadas erroneamente como *multi-tenant*) e Falso Negativo (taxa de instâncias *multi-tenant* que foram incorretamente classificadas como *Normais*).

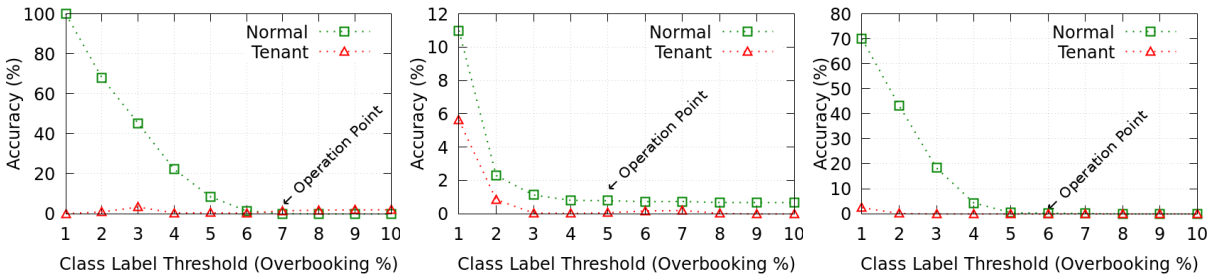
A avaliação (Figura 31) demonstra que o modelo de auditoria em duas camadas proposto classifica corretamente os nós *Normais* e *Multi-tenant* para todas as topologias testadas. Em relação à topologia para consumo de CPU (WC), o modelo classificou quando o nó apresentou taxas de *overbooking* maiores que 6% (Figura 31.a) e 7% (Figura 31.d), enquanto apresentou taxas de Falso Positivo de apenas 0,05% e 0,01%, e taxas de Falso Negativo de 0,81% e 0,43% para NB e DT respectivamente. Igualmente ocorre para as topologias ligadas a disco (WCF) e a rede (TT) onde as taxas de *overbooking* encontraram-se superiores a 7% para o classificador NB e 5% e 6% para o classificador DT, respectivamente.

Adicionalmente, é possível observar que a acurácia é relativa à limiar do *overbooking*, uma vez que o desempenho da aplicação e a utilização de diferentes recursos se torna mais

significativa. A interferência mínima necessária para cada classificador e topologia é marcada como *Operation Point* na Figura 31, e foi escolhida quando as taxas de FP e FN atingem valor inferior a 1%.



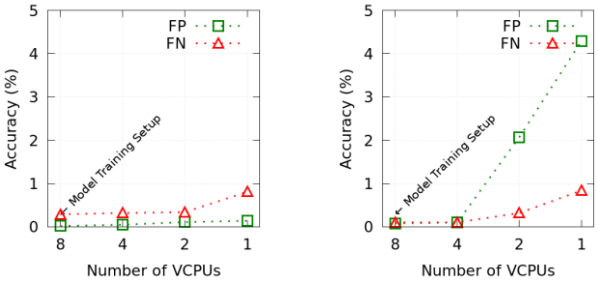
(a) NB for CPU-bound topology (8 vCores) (b) NB for disk-bound topology (c) NB for network-bound topology



(d) DT for CPU-bound topology (8 vCores) (e) DT for disk-bound topology (f) DT for network-bound topology

Figura 31 – Avaliação classificadores

Para responder à questão (ii), foi realizada uma avaliação sobre a topologia que consume de CPU (WC), com diferentes configurações de máquina virtual, variando de 8 a 1 vCPUs. O treinamento e a avaliação ocorreu no ambiente com 8 vCPUs, nessa e nas demais configurações (Figura 32). O classificador NB supera o classificador DT quando uma configuração de *hardware* diferente é considerada. As taxas de FP e FN aumentam de acordo com a diferença entre ambiente de treinamento e a configuração de *hardware* avaliada. Essa avaliação permite observar que o modelo de auditoria proposto é capaz de realizar a detecção em diferentes configurações de *hardware*, e quando uma configuração próxima do ambiente de treinamento as taxas de detecção permanecem semelhantes.

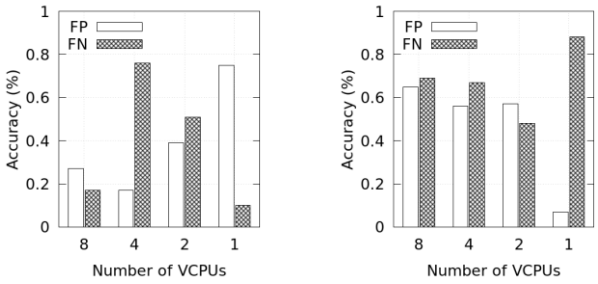


(a) NB para diferentes configuracoes de hardware

(b) DT para diferentes configurações de hardware

Figura 32 – Acurácia de auditoria em nuvem privada.

Para responder a questão (iii), foi avaliada a topologia de consumo de CPU (WC) na nuvem pública da Amazon AWS [Amazon AWS, 2006]. A fim de comprovar se os eventos rotulados estavam corretos, foi utilizado a métrica de "roubo" de CPU (*Steal Time*) fornecida pela Amazon. A Figura 33 apresenta o desempenho do modelo de auditoria no ambiente de nuvem pública com aplicação do modelo treinado em nuvem privada, com diferentes configurações de *hardware.*



(a) NB para diferentes configurações de hardware em um ambiente de nuvem pública.

(b) DT para diferentes configurações de hardware em um ambiente de nuvem pública.

Figura 33 - Acurácia de auditoria em nuvem pública.

O modelo proposto foi capaz de generalizar o comportamento da aplicação, as taxas de FP e FN foram inferiores a 1% para os classificadores avaliados em todas as configurações de *hardware* consideradas. As avaliações permitem concluir que o modelo de auditoria de duas camadas proposto é capaz de detectar quando uma aplicação enfrenta interferências *multi-tenant*, ou seja, o valor de *overbooking* é superior a 6% em nuvens públicas e privadas. Adicionalmente, o administrador do sistema pode treinar no mecanismo proposto em nuvem

privada com a configuração de *hardware* diferente e monitorar adequadamente sua aplicação em nuvem pública.

Monitorar recursos virtuais ou desempenho da aplicação de maneira isolada não permite ao cliente aferir com precisão se a queda de desempenho ocorre pelo aumento de demanda por parte da aplicação ou se é por alguma eventual falha por parte do provedor de nuvem. A avaliação do modelo de auditoria em duas camadas proposto, utilizando *Apache Storm* como estudo de caso, demonstra a viabilidade da solução. O modelo, além de identificar interferência *multi-tenant*, permite o seu treinamento por parte do administrador do sistema em ambiente controlado em nuvem privada e sua generalização para aplicação em nuvem pública.

5.7. Discussão do Capítulo

As avaliações relatadas nesse capítulo contextualizam a importância de métodos de escalonamento que avaliam o estado físico do *hardware* em que a máquina virtual encontra-se em execução, a fim de reduzir os impactos de desempenho sobre as aplicações. Ambientes *multi-tenant* são uma realidade, principalmente no ambiente de nuvem computacional (e.g. Amazon, Microsoft Azure, etc.) e podem influenciar de forma significativa o desempenho de aplicações em sistemas de processamento para *Big Data* baseados em *stream*, como o *framework Apache Storm*, por exemplo.

Identificar com transparência o compartilhamento de recursos em provedores de computação em nuvem não é uma tarefa trivial, uma vez que os provedores de serviço tipicamente não fornecem informações referentes ao estado físico do *hardware* que executa a máquina virtual. Diante disso, entende-se a real necessidade de que os algoritmos de escalonamento se utilizem de políticas que avaliem e considerem o estado físico do *hardware* de forma a evitar a degradação do desempenho das aplicações.

A abordagem proposta nesta tese objetivou prover uma solução que considere a interferência *multi-tenant*. Para tanto, a solução atua em três níveis distintos. Primeiramente, uma abordagem de escalonamento e reescalonamento de tarefas em nível de *cluster* foi desenvolvida e avaliada. A abordagem proposta permitiu a identificação de *tenants* sob interferência *multi-tenant* e a correta distribuição das tarefas de acordo com o estado físico de cada *tenant*. Posteriormente, uma abordagem de provisionamento de recursos foi proposta e

avaliada. Recursos computacionais adicionais foram provisionados e encerrados de acordo com a necessidade de processamento da infraestrutura como um todo, considerando tanto o estado lógico quanto físico do mesmo. Por fim, uma abordagem de balanceamento de carga, entre os recursos adicionais, foi avaliada, demonstrando a viabilidade da proposta.

Finalmente, alternativas ao *microbenchmark* foram propostas e avaliadas e demonstrou-se que, tanto a abordagem baseada em aprendizagem de máquina quanto a abordagem baseada em análise de fluxo permitem a correta identificação de interferências *multi-tenant*.

Capítulo 6

Considerações Finais

Esta tese de doutorado avaliou o impacto *multi-tenant* em nuvem computacional, através do *framework* de *big data* para processamento de *data stream*, *Apache Storm*. As análises discutidas evidenciaram que o fato de não considerar o estado físico dos nós que hospedam as máquinas virtuais é uma prática ineficiente no escopo estudado. Uma vez que, por se tratar de nuvem computacional, os recursos físicos são compartilhados e podem ter sido esgotados por outros clientes (inquilinos) da nuvem.

Para tratar a questão do compartilhamento de recursos, foi implementado o *Dynamic Scheduler* – DySc. Essa solução avalia de forma contínua o estado dos recursos físicos disponíveis no ambiente de computação em nuvem através de *microbenchmarks*. Essas medidas foram vinculadas a políticas de escalonamento e reescalonamento, viabilizando evitar a distribuição das tarefas para nós com recursos computacionais exaustos. As avaliações experimentais realizadas com a solução DySc, quando comparadas com o escalonador padrão do *framework Apache Storm*, evidenciaram a viabilidade de otimização de desempenho de aplicações em 50% para CPU, 62% para disco e 43% para rede, quando o mesmo recurso encontrava-se sendo consumido por inquilinos externos no cenário *Multi-Tenant Cluster*.

O contexto de processamento de *stream* em *big data* é extremamente variável, mesmo com uma política eficiente de escalonamento, como a solução DySc que considera a disponibilidade de recursos físicos e virtuais, pois ainda assim podem ocorrer situações onde existam aumentos significativos de demanda computacional. Isso implica numa estratégia de provisionamento de recursos que considere as características de uma nuvem computacional *multi-tenant*.

Para esse fim, foram implementadas duas soluções intituladas *Dynamic Load Balancing* - DyLB e *Elastic Resource Provisioning* – ERPr. Com objetivo de fornecer

provisionamento de recursos em nuvem, o ERPr inicia e finaliza *clusters* computacionais, enquanto o DyLB fornece a política de balanceamento de carga entre as nuvens em operação. A implementação das referidas soluções operam baseadas na política NFV (*Network functions Virtualization*). Dessa forma, não necessitam de um equipamento dedicado, e podem ser instanciados em qualquer local da rede. As avaliações revelaram que essas soluções quando aplicadas no contexto da pesquisa redirecionaram 72% da carga computacional para o *cluster* que dispunha a maior quantidade de recursos computacionais livres, e de modo consequente obteve um ganho de 22% em relação ao tradicional algoritmo *round-robin*, tipicamente utilizado nos sistemas de balanceamento de carga.

De maneira a complementar o trabalho e com objetivo de atenuar o consumo extra de recursos provocado pelos *microbenchmarks*, a seção 5.6 avalia alternativas na identificação de interferência *multi-tenant*. Duas alternativas foram desenvolvidas e avaliadas. A primeira conta com o SDN para avaliar os fluxos de rede via controlador na identificação de VMs que estão sob interferência e os resultados apontaram que é possível identificar a variação de desempenho empregando o método em questão. A segunda solução fez uso de aprendizagem de máquina e igualmente apresentou resultados que atestam sua viabilidade na identificação de interferência sob o mesmo *hardware*. O modelo de auditoria ocorreu em dois níveis e concatenou aprendizagem de máquina e métricas da aplicação para treinamento da base.

Considerando os resultados obtidos na presente pesquisa, identificou-se a possibilidade de trabalhos futuros. Dentre eles destacam-se:

- A utilização do provisionamento de recursos de maneira horizontal, porém em granularidade de *tenant*, em oposto à abordagem baseada em *cluster*. Como vantagem, esta abordagem permitiria o aumento da capacidade computacional do *cluster*, sem a necessidade de replicar a infraestrutura. Adicionalmente, uma abordagem baseada em SDN seria responsável pela mudança dos fluxos de maneira transparente a aplicação. Assim, evita-se mudanças na aplicação para adicionar o novo *tenant* na infraestrutura lógica;
- Provisionamento de recursos em nível de executor no *Apache Storm*. Esta abordagem provém recursos adicionais ao cluster considerando a sobrecarga em cada tipo de executor, por exemplo, *spout* ou *bolt*. Dessa maneira a sobrecarga de processamento

poderia ser tratada de acordo com seu tipo, seja na geração dos dados ou no seu processamento;

- Distribuição de carga a nível de rede em cenários em que o cluster atual é capaz de processar a demanda. Essa abordagem permitiria a redistribuição da carga durante períodos de multi-tenant em *tenants* específicos. Dessa maneira, evita-se que novos *tenants* sejam criados caso a infraestrutura atual seja capaz de processar a carga gerada ao longo do tempo. Adicionalmente, evita-se que os executores sejam terminados em tenants com impacto multi-tenant, diminuindo assim o impacto da proposta.

6.1. Publicações

- O trabalho completo com emprego de *microbenchmarks* para detecção de variação de desempenho *multi-tenant* encontra-se sob revisão (*Major Review*) no ***Journal of Network and Computer Applications* (*JNCA, Qualis A2*).** Os revisores sugeriram algumas modificações no documento que já foram realizadas e encontram-se em análise.
- A alternativa que emprega Aprendizagem de Máquina (sub-seção 5.6.2) foi aceito para publicação no ***18th Annual IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGrid 2018 – Qualis A1, trilha principal)*.**

Referências Bibliográficas

[Agrawal et al. 2011] Agrawal, D., Das, S., El Abbadi, A. Big data and cloud computing: current state and future opportunities. In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT ’11, pp 530–533, New York, NY, USA. ACM.

[Akidau, T., 2013] Tyler Akidau, Alex Balikov, Kaya Bekiroglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle: MillWheel: fault-tolerant stream processing at internet scale. Proc. VLDB Endow. 6, 11 (2013), pp 1033-1044. DOI: http://dx.doi.org/10.14778/2536222.2536229

[Alice’s Adventures in Wonderland, 2008] Alice’s Adventures in Wonderland. [Online] Disponível em: http://www.gutenberg.org/files/11/11-pdf.

[Amazon AWS, 2006] Amazon AWS [Online]. Disponível em: aws.amazon.com [Acesso em: Outubro 2017]

[Ananthanarayanan, R. et al. 2013] R. Ananthanarayanan, V. Basker, S. Das, A. Gupta, H. Jiang, T. Qiu, A. Reznichenko, D. Ryabkov, M. Singh, S. Venkataraman. Photon: fault-tolerant and scalable joining of continuous data streams. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13). pp 577-588. DOI: http://dx.doi.org/10.1145/2463676.2465272

[Aniello, L., et al, 2013] Aniello, L., Baldoni, R. and Querzoni, L. (2013). Adaptive online scheduling in storm. In Proceedings of the 7th ACM international conference on Distributed event-based systems (DEBS '13). ACM, pp 207-218. DOI=http://dx.doi.org/10.1145/2488222.2488267.

[Apache, 2016]. Apache Hadoop. [Online] Disponível em: hadoop.apache.org/docs/current/. [Acesso em: Outubro 2017].

[B. Han,. Et al. 2015] B. Han, V. Gopalakrishnan, L. Ji and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," in IEEE Communications Magazine, vol. 53, no. 2, pp. 90-97, doi: 10.1109/MCOM.2015.7045396.

[Baset, S. A., et al, 2012] Baset, S. A., Wang, L. and Tang, C., “Towards an understanding of oversubscription in cloud,” in Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. USENIX Association.

[Bohn, C. A. e Lamont, G. B., 2002] Bohn, C. A. and Lamont, G. B. Load balancing for heterogeneous clusters of PCs. Future Generation Computer Systems, v. 18, n. 3, pp. 389–400. doi: 10.1016/S0167-739X(01)00058-9

[Bouchenak, S., et al, 2013] Bouchenak, S., Chockler, G., Chockler, H., Gheorghe, G., Santos, N., Shraer, A. 2013. Verifying cloud services: present and future. ACM SIGOPS Operating Systems Review, v. 47, n. 2, pp. 6–19.

[C.S., 2016] Capacity Scheduler. [Online] Diponível em: hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html [Acesso em: Janeiro 2016]

[Chen, M. et al., 2014] Chen, M., Mao, S., & Liu, Y. Big data: A survey. Mobile Networks and Applications, v. 19, n. 2, p. 171-209. doi.org/10.1007/s11036-013-0489-0

[Collectl, 2005] Collectl [Online]. Disponível em: collectl.sourceforge.net/ [Acesso em: Outubro 2017]

[Das A., et al, 2013] Das A., Lumezanu C., Zhang Y., Singh V., Jiang G., Yu C. Transparent and Flexible Network Management for Big Data Processing in the Cloud. USENIX Workshop on Hot Topics in Cloud Computing.

[Dean, J. and Ghemawat, S., 2008] Dean, J.; Ghemawat, S. Mapreduce: simplified data processing on large clusters. Communications of the ACM, ACM, v. 51, n. 1, pp. 107–113.

[Erickson, D., 2013] Erickson, D., “The Beacon OpenFlow controller,” in Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ser. HotSDN ’13. ACM, 2013, pp. 13–18.

[ETSI, 2012] ETSI (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action. White paper, European Telecommunications Standards Institute (ETSI). portal.etsi.org/NFV/NFV_White_Paper.pdf

[F.S., 2016] Fair Scheduler. [Online] Disponível em: hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/FairScheduler.html [Acesso em: Janeiro 2016]

[Feng, T., et al. 2015] Feng, T., Zhuang, Z., Pan, Y. and Ramachandra, H. A Memory Capacity Model for High Performing Data-filtering Applications in Samza Framework. In: Big Data (Big Data), 2015 IEEE International Conference on. IEEE, 2015. pp. 2600-2605.

[Ferguson, A., 2013] Ferguson, A., Guha, a and Liang, C. (2013). Participatory networking: An API for application control of SDNs. In: ACM SIGCOMM computer communication review. ACM, 2013. pp. 327-338.

[Floodlight, 2012] Floodlight is a Java-based OpenFlow controller, 2012. [Online]. Disponível em: floodlight.openflowhub.org [Acesso em: Janeiro 2018]

[Galante, G., et al, 2012] Galante, G., Bona, L. C. E., Rego, P. A. L. and Souza, J. N. ERHA: Execution and Resources Homogenization Architecture. In Proc. of the Cloud Computing, pp. 265.

[Gantz e Reinsel, 2011] Gantz, J. e Reinsel, D. (2011). Extracting value from chaos. IDC iview, v. 1142, n. 2011, pp. 1-12.

[GENI, 2006] Peterson, L., Anderson, T., Blumenthal, D., GENI design principles. Computer, v. 39, n. 9, p. 102–105, 2006

[Grandl, R., et al, 2014] Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S. and Akella, A. Multi-resource packing for cluster schedulers. ACM SIGCOMM Computer Communication Review, v. 44, n. 4, pp. 455–466.

[Grobauer et al. 2010] Grobauer, B., Walloschek, T., e Stöcker, E. Understanding CloudComputing Vulnerabilities. IEEE Security & Privacy, v. 9, n. 2, pp. 50-57.

[Gude, N., et al., 2008] Gude, N., Koponen, T., Pettit, J., Pfaff, J., Casado, M., McKeown, N., Shenker, S. NOX: towards an operating system for networks. *SIGCOMM Computer Communication Review*, v. 38, n. 3, pp. 105–110. doi: 10.1145/1384609.1384625

[Han, Z. and Zhang, Y., 2015] Han, Z. and Zhang, Y. (2015). Spark: A Big Data Processing Platform Based on Memory Computing. In: Parallel Architectures, Algorithms and Programming (PAAP), 2015 Seventh International Symposium on. IEEE, 2015. pp. 172-176.

[Handigol, N., et al, 2009] Handigol, N., Seetharaman, S., Flajslik, M., McKeown, N. and Johari, R. Plug-n-serve: Load-balancing web traffic using OpenFlow. ACM Sigcomm Demo, v. 4, n. 5, pp. 6..

[Handigol, N., et al, 2010] Handigol, N., Seetharaman, S., Flajslik, M. and Gember, A. Aster * x : Load-Balancing Web Traffic over Wide-Area Networks.

[Hashem, I. A. T., et al, 2015] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., et al. (2015). The rise of "big data" on cloud computing: Review and open research issues. Information Systems, v. 47, pp. 98–115.

[He, S., et al, 2012] He, S., Guo, L., Ghanem, M. and Guo, Y. Improving Resource Utilisation in the Cloud Environment Using Multivariate Probabilistic Models. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE. pp. 574-581.

[HPE, 2014] HPE Helion Eucalyptus [Online] Disponível em: http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html [Acesso em: Setembro 2017]

[Hu, et al. 2014] Hu, H., Wen, Y., Chua, T. S., Li, X. Toward scalable systems for big data analytics: A technology tutorial. Access, IEEE, v. 2, pp. 652–687.

[Hwang, K., et al., 2015] Hwang, K., Bai, X., Shi, Y. Cloud Performance Modeling and Benchmark Evaluation of Elastic Scaling Strategies. IEEE Transactions on parallel and distributed systems, v. 27, n. 1, p. 130-143.

[iPerf, 2003] iPerf - The network bandwidth measurement tool. [Online] Disponível em: https://iperf.fr/ [Acesso em: Setembro 2017]

[Isard M. et al. 2007] Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D. Dryad: dis- tributed data-parallel programs from sequential building blocks. In: ACM SIGOPS operating systems review. p. 59-72.

[Isard, M., et al, 2009] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K. and Goldberg, A. Quincy: Fair scheduling for distributed computing clusters. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. pp. 261-276.

[JAX-RS, 2007] Java API for RESTful Services. [Online]. Disponível em: https://jax-rsspec.java.net/. [Acesso em: Janeiro 2017]

[Kaur, S. et al., 2014] Kaur, S., Singh, J. and Ghumman, N. S. (2014). Network Programmability Using POX Controller. *International Conference on Communication, Computing & Systems (ICCCS)*, pp. 134.

[Kreutz, D., et al, 2015] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. Software-defined networking: A comprehensive survey. Proceedings of the IEEE, v. 103, n. 1, p. 14–76.

[Lara, A., et al. 2013] Lara, A., Kolasani, A., Ramamurthy, B. Network Innovation using OpenFlow: A Survey. IEEE communications surveys & tutorials, v. 16, n. 1, pp. 493-512.

[Malewicz G, et al. 2010] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., Czajkowski, G. (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACMSIGMOD international conference on management of data. ACM, pp 135–146.

[Medved, J., et al. 2014] Medved, J., Varga, R., Tkacik, A. and Gray, K. (2014). OpenDaylight: Towards a model-driven SDN controller architecture. In: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a. IEEE. p. 1-6.

[Mijumbi, R., et al. 2016] Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., De Turck, F. and Boutaba, R., "Network Function Virtualization: State-of-the-Art and Research Challenges," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262. doi: 10.1109/COMST.2015.2477041

[N. McKeown, et al. 2008] 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.

[ONF, 2014] Open Networking Foundation (ONF), 2014. SDN architecture 1.0, [Online]. Disponível em: opennetworking.org/

[OpenDaylight, 2013] “OpenDaylight: A Linux Foundation Collaborative Project,” 2013. [Online]. Disponível em: opendaylight.org

[Paniagua, D., 2016] *Amazon Web Services* [Mensagem Pessoal]. Mensagem recebida de: <aws-cs-latam@amazon.com> 15 Jun. 2016.

[Peng, B., et al, 2015] Peng, B., Hosseini, M., Hong, Z., Farivar, R. and Campbell, R. (2015). R-Storm: Resource-Aware Scheduling in Storm. In: Proceedings of the 16th Annual Middleware Conference. ACM. pp. 149-161.

[Poddar, R., et al, 2015] Poddar, R., Vishnoi, A. and Mann, V. HAVEN : Holistic Load Balancing and Auto Scaling in the Cloud. In: Communication Systems and Networks (COMSNETS), 2015 7th International Conference on. IEEE, 2015. pp. 1-8. DOI: 10.1109/COMSNETS.2015.7098681.

[PROC, 1994] Process information pseudo-file system. [Online] Disponível em: http://linux.die.net/man/5/proc [Acesso em: Março 2017]

[Rego, P. A. L., et, al., 2011] Rego, P. A. L., Coutinho, E. F., Gomes, D. G. and De Souza, J. N. (2011). FairCPU: Architecture for allocation of virtual machines using processing features. In: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on. IEEE. p. 371-376.

[Rosenblum, M. e Garfinkel, T., 2005] Rosenblum, M. and Garfinkel, T. "Virtual machine monitors: Current technology and future trends," IEEE Computer, vol. 38, no. 5, pp. 39–47.

[S4, 2013] S4. 2013. Distributed stream computing platform. [Online]. Disponível em: incubator.apache.org/s4/

[Schad, J. et al, 2010] Schad, J., Dittrich, J. and Quiané-Ruiz, J.-A. (2010). Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. Proceedings of the VLDB Endowment, v. 3, n. 1-2, p. 460–471.

[Segalin, D. et al, 2015] Segalin, D., Santin, A. O., Marynowski, J. E. and Maziero, C. (2015). An Approach to Deal with Processing Surges in Cloud Computing. In: Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. IEEE, 2015. pp. 897-905.

[Shen, Z., et al, 2011] Shen, Z., Subbiah, S., Gu, X. and Wilkes, J. (2011). CloudScale: elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. pp. 5.

[Singh, A., et al, 2008] Singh, A., Korupolu, M. and Mohapatra, D. Server-storage virtualization: Integration and load balancing in data centers. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008. pp. 53.

[Smith e Nair 2005] Smith, J. E. e Nair, R. (2005). The architecture of virtual machines. IEEE Computer, v. 38, n. 5, p. 32-38.

[Spotify, 2016] Spotify. 2016. [Online]. Disponível em: spotify.com

[Stonebraker et al. 2005] Stonebraker, M.; Çetintemel, U.; Zdonik, S. The 8 requirements of real-time stream processing. ACM SIGMOD Record, ACM, v. 34, n. 4, pp. 42–47.

[Storm Throughput Test, 2012] Storm Throughput Test, 2012. [Online]. Disponível em: https://github.com/stormprocessor/storm-benchmark/blob/master/src/jvm/storm/benchmark/ThroughputTest.java

[Storm UI REST, 2014] Storm UI REST, 2014. [Online]. Available: https://github.com/Parth-Brahmbhatt/incubator-storm/blob/master/STORM-UI-REST-API.md

[Storm, 2016] Apache Storm. 2016. [Online]. Disponível em: storm-project.net/

[Sysbench, 2009] Sysbench - Modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters. [Online] Disponível em: https://launchpad.net/sysbench [Online] [Acesso em: Setembro 2017]

[Tomas, L. e Tordsson, J., 2014] Tomas, L. and Tordsson, J., "An Autonomic Approach to Risk-Aware Data Center Overbooking," IEEE Trans. Cloud Comput., vol. 2, no. 3, pp. 292–305.

[Toshniwal, A. et al. 2014] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S. & Bhagat, N. (2014). "Storm@ twitter". In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, pp. 147-156. ACM.

[Twitter record, 2013] Twitter. New Tweets per second record, and how. Twitter Engineering Blog, v. 16, 2013. Disponível em: blog.twitter.com/2013/new-tweets-per-second-record-and-how

[Twitter, 2016] Twitter. 2016. [Online]. Disponível em: twitter.com/

[Vecchiola, C., et al, 2012] Vecchiola, C., Calheiros, R.N., Karunamoorthy, D., Buyya , R. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. Future Generation Computer Systems, v. 28, n. 1, p. 58-65.

[Verma, A., et al, 2011] Verma, A., Cherkasova, L. and Campbell, R. H. Resource provisioning framework for MapReduce jobs with performance goals. In: Proceedings of the 12th International Middleware Conference. International Federation for Information Processing, pp. 160-179.

[VMware, 2016] VMware virtualization. [Online] Disponível em: www.vmware.com

[Wang, G., et al, 2012] Wang, G., Ng, T. S. E. and Shaikh, A. Programming your network at run-time for big data applications. In: Proceedings of the first workshop on Hot topics in software defined networks. pp. 103-108.

[Wang, R., et al, 2011] Wang, R., Butnariu, D. and Rexford, J. OpenFlow-Based Server Load Balancing Gone Wild. Hot-ICE'11 Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and servicesworks and services. v. 11, pp. 12.

[Weka, 2006] Weka [Online]. Disponível em: weka.sourceforge.net [Acesso em: Outubro 2017]

[Word Count Topology, 2013] Word Count Topology, 2013. [Online]. Disponível em: https://github.com/apache/storm/blob/master/examples/storm-starter/src/jvm/storm/starter/WordCountTopology.java

[Xen Project, 2016]. Xen Project [Online]. Disponível em: xenproject.org/

[Xu, J., et al, 2014] Xu, J., Chen, Z., Tang, J. and Su, S. T-Storm: Traffic-Aware Online Scheduling in Storm. In: Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on. IEEE. pp. 535-544.

[Yahoo Japan, 2016] Yahoo Japan. 2016. [Online]. Disponível em: yahoo.co.jp/

[Yahoo, 2016] Yahoo. 2016. [Online]. Disponível em: yahoo.com/

[Yan, F., et al, 2015] Yan, F., Cherkasova, L., Zhang, Z. and Smirni, E. (2015). DyScale: a MapReduce Job Scheduler for Heterogeneous Multicore Processors. IEEE Transactions on Cloud Computing, v. 5, n. 2, p. 317-330.

[YARN, 2016] Apache Hadoop YARN. [Online] Disponível em: hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[Zhang, Z. et al. 2010] Zhang, Z., Gu, Y., Ye, F., et al. (2010). A Hybrid Approach to High Availability in Stream Processing Systems. IEEE 30th International Conference on Distributed Computing Systems (ICDCS), pp. 138–148.

[ZooKeeper, 2016] Apache Zookeeper. 2016. [Online]. Disponível em: zookeeper.apache.org

images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.