Arcabouço para Orquestração Osmótica de *Cloud Slices*

Aryadne G. P. Rezende[1], Gustavo S. Marques[1], Raquel F. Q. Lafetá[1], Rafael Pasquini[1]

[1]Programa de Pós-Graduação em Computação UFU
Universidade Federal de Uberlândia (UFU) – Minas Gerais – MG – Brasil

{aryadne.guardieiro, guto_silveira_}@ufu.br

{raquel.lafeta, rafael.pasquini}@ufu.br

Abstract. *This research builds upon the Slice as a Service paradigm proposed in the Novel Enablers for Cloud Slicing (NECOS) project. Assuming the provision of end-to-end slices which are composed by resources coming from multiple infrastructure providers, this work explores an osmotic orchestration approach in a cloud slice context, aiming to preserve SLAs during run time. The objective is to develop a mechanism to orchestrate the rearrangement of spare resources placed in a lightly loaded slice part towards a heavily loaded one. This paper brings an innitial experiment to validate the operation of the built testbed.*

Resumo. *Essa pesquisa baseia-se no paradigma Slice as a Service proposto no projeto Novel Enablers for Cloud Slicing (NECOS). Assumindo o fornecimento de slices fim-a-fim, as quais são compostas por recursos vindos de multiplos provedores de infraestrutura, esse trabalho explora uma abordagem de orquestração osmótica, em um contexto de cloud slice, visando preservar o SLA durante o tempo de execução. O objetivo é desenvolver um mecanismo para orquestrar o rearranjo de recursos excedentes contidos em uma parte da slice pouco utilizada para uma parte que esteja mais sobrecarregada. Este artigo traz um experimento inicial para validar a operação do ambiente de testes construído.*

1. Introdução

A Computação em Nuvem (*Cloud Computing*) é o paradigma que emergiu da oferta de recursos computacionais na forma de um modelo *pay-as-you-go*, no qual o cálculo do pagamento pelo serviço é feito conforme seu uso em um determinado período. Um dos diferenciais da Computação em Nuvem é a elasticidade, a qual permite aos usuários adquirir ou liberar a quantidade necessária de recursos computacionais, de acordo com sua necessidade [Netto et al. 2014].

O modelo de negócio da Computação em Nuvem oferece serviços em três níveis: infraestrutura (*IaaS - Infrastructure as a Service*), plataforma (*PaaS - Platform as a Service*) e software (*SaaS - Software as a Service*), sendo que existem diversos provedores disponibilizando seus recursos em cada camada. Ortogonal a todas essas camadas está o conceito de *Slice* ou *Cloud Slice*. Uma *slice*, ou fatia, pode ser vista como um conjunto de recursos físicos ou virtuais (rede, processamento e armazenamento) que podem acomodar componentes de um serviço, de maneira integrada e independente de outras *slices* [Freitas et al. 2018]. O enorme número de combinações possíveis e muitas vezes a complexidade de integração desses recursos, tornam difícil o trabalho de organizações que

queiram oferecer seus serviços utilizando diferentes provedores de computação, armazenamento e conectividade. Toda essa complexidade torna o lançamento de novos serviços uma tarefa demorada e cara [Silva et al. 2018].

Com a finalidade diminuir essa complexidade, surgiu o projeto NECOS (*Novel Enablers for Cloud Slicing*), no qual se insere este trabalho [NECOS 2018b]. O objetivo geral do projeto é investigar um novo modelo de negócio chamado de *Slice as a Service*. Neste contexto, será testada a criação de *cloud slices* sob demanda, em um ambiente multi-domínio. Além da criação, será investigada a reconfiguração automática de recursos da *slice*, a fim de garantir a qualidade de serviço dos clientes do NECOS para com seus usuários finais.

Primeiro, o cliente fornecerá uma descrição da *slice* que irá precisar para o sistema NECOS. Depois de alocado esse conjunto de recursos, vindos de um ou mais provedores, o cliente poderá prover seus serviços ao seus usuários finais. Devido a natureza dinâmica da maioria das aplicações hospedadas na nuvem, a demanda do cliente poderá sofrer variações para mais ou para menos. Por essa razão, serão feitas constantes verificações do *Service Level Agreement* (SLA) que está sendo provido aos usuários finais do cliente (mais detalhes dessa estimativa em [Pasquini and Stadler 2017]). Buscando evitar quebras nesse acordo de serviço, o sistema NECOS ajustará as configurações da *slice*, objetivando manter a qualidade do serviço oferecido. Para manter o SLA contratado e os custos do cliente proporcionais a demanda de maneira dinâmica, é necessário um orquestrador inteligente que gerencie a alocação e desalocação de recursos.

Nesse cenário, o foco deste trabalho é o redimensionamento inteligente e automático de *slices*. A abordagem escolhida para a tratativa desse problema é a Computação Osmótica [Villari et al. 2016]. Como na biologia, em que a osmose se refere a passagem de água de um meio a outro para igualar concentrações de solutos, recursos também podem fluir para onde a demanda se concentra, evitando desperdícios e provendo melhor qualidade ao cliente final. Assim, o objetivo é avaliar o uso da Computação Osmótica no contexto da orquestração de *cloud slices*. Adiante, é proposto um arcabouço para orquestração osmótica de *cloud slices*, que adapta os recursos das partes da *slice* de maneira osmótica, com a finalidade de manter o SLA acordado aproximando o custo operacional ao mais baixo possível. Neste artigo, apresentamos também resultados de um experimento inicial, no qual o ambiente de testes desenvolvido coleta métricas de uma aplicação de vídeo sob demanda sendo ofertada em um *slice*. A partir destas métricas o orquestrador irá estimar a situação do SLA, ou seja, o primeiro passo na construção deste trabalho.

Na próxima Seção, são apresentados trabalhos que tratam da orquestração de recursos, no contexto de nuvem, IoT e serviços, intrinsecamente relacionados com este trabalho. Após, a Seção 3 expõe um arcabouço para orquestração de *cloud slices* de maneira osmótica, no contexto do projeto NECOS. Na sequência, a Seção 4 mostra a plataforma de experimentação elaborada para a investigação desse orquestrador. Em seguida, a Seção 5 mostra os resultados iniciais nessa plataforma. Por fim, a Seção 6 reúne as considerações finais juntamente com os próximos passos desta pesquisa.

2. Trabalhos Relacionados e Fundamentação Teórica

O problema da orquestração de recursos pode ser observado de diferentes ângulos, dependendo dos recursos que se planeja gerenciar. Os trabalhos citados em [Qu et al. 2018], por exemplo, lidam com a orquestração de máquinas virtuais. Em [Casalicchio and Perciballi 2017] e [Casalicchio 2019] analisa-se a orquestração a nível de contêineres. Já em [Carnevale et al. 2018] é descrito um orquestrador osmótico que gerencia dispositivos no contexto IoT (*Internet of Things*). Por outro lado, o trabalho [Sciancalepore et al. 2017] ilustra um orquestrador de *slices* em um contexto 5G.

Além de abordagens de orquestração, também estão relacionados a este artigo os temas: Computação Osmótica [Villari et al. 2016] e estimação de QoS (*Quality of Service*) a partir de métricas do provedor de serviço [Pasquini and Stadler 2017]. Adiante, nos aprofundaremos em conceitos importantes extraídos dessas publicações e em como elas se relacionam a este trabalho.

Em [Qu et al. 2018] é introduzido um tema pertinente na orquestração em nuvem: o *auto-scaling* (dimensionamento automático) de recursos. Dada a possibilidade de redimensionar facilmente os recursos virtuais na nuvem, originou-se o desafio de prover apenas a quantidade necessária de recursos para determinada aplicação em um dado momento. Na *survey*, é feita uma compilação das mais recentes pesquisas sobre o tema em ambientes de nuvem, focando no dimensionamento automático de *Virtual Machines* (VMs). Os trabalhos são classificados quanto a como são abordados os desafios do ciclo MAPE (*Monitoring, Analysis, Planning, and Execution*) [Kephart and Chess 2003]. As etapas desse ciclo, ilustradas na Figura 1, podem ser descritas como:

Monitoramento: nesta etapa são colhidos indicadores de performance, que serão usados para determinar estratégias de dimensionamento. O primeiro desafio dessa etapa é selecionar quais são os indicadores de performance mais significativos, o que aumenta a precisão da estimação dos recursos necessários e diminui os custos de tráfego de informação na rede. Em segundo lugar, é importante definir um intervalo de monitoria, o que tem impacto no fluxo de informação na rede e também influencia no quão sensível às mudanças será o orquestrador.

Análise: nesta fase, o orquestrador determina, por meio dos dados coletados, se é necessário executar alguma alteração nos recursos já alocados. Primeiramente, é preciso definir o momento de dimensionamento, que pode ser proativo ou reativo a mudanças. Se o método for proativo, é preciso escolher estratégias de estimação de carga. Também é preciso avaliar se o sistema suportará adaptação à mudanças, ou será composto de regras fixas. Outro ponto relevante refere-se a mitigar a oscilação na distribuição dos recursos, pois redimensionamentos sucessivos em um curto espaço de tempo ocasionam desperdícios e perdas de SLA.

Planejamento: neste ponto, é feita a estimativa de recursos que devem ser providos ou desprovidos na próxima ação de redimensionamento. É importante que seja selecionada a quantidade certa para reduzir custos financeiros. As dificuldades desse estágio incluem estimar quais recursos e em quais quantidades serão necessários para a situação atual ou futura dos serviços e como combiná-los. Os desafios dessa etapa a tornam um problema de otimização com um grande espaço de possibilidades, sendo um problema NP-hard a geração de um plano de provisão perfeito.

Execução: no fim do ciclo MAPE, a execução será a aplicação do plano de provisão produzido na etapa anterior, executando-se ações de redimensionamento por meio de

API's (*Application Programming Interface*) disponibilizadas pelos provedores de serviço. Quando se trata de aplicações que podem ser implantadas em diferentes *data centers*, ou caso ações em camadas distintas precisem ser executadas, é necessário lidar com diferentes API's dependendo dos provedores de serviço envolvidos, o que aumenta o grau de dificuldade de se construir um orquestrador genérico.

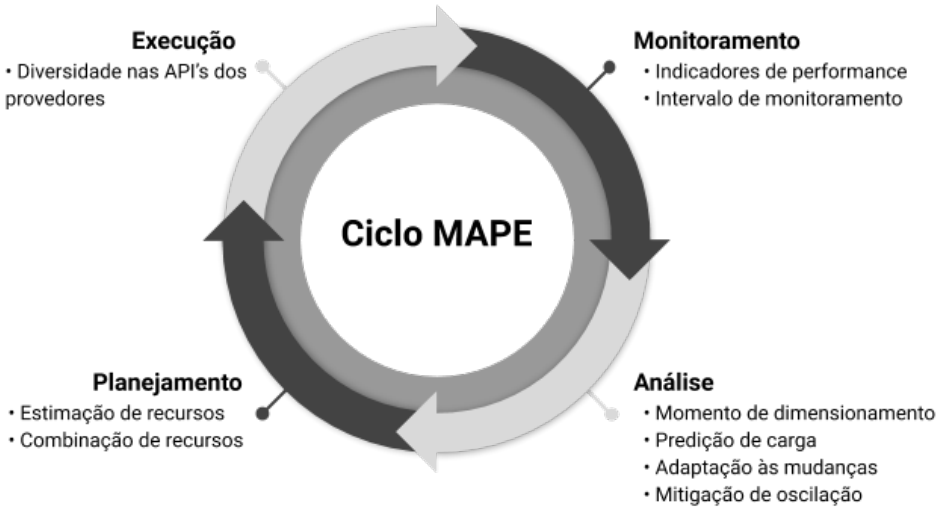

Figura 1. Ciclo MAPE e os principais desafios de cada etapa

Além do ciclo de orquestração, constam em [Qu et al. 2018] quatro operações diferentes de dimensionamento. No que se refere ao aumento de recursos, a operação *scaling up* aumenta os recursos internos de uma máquina virtual (número de CPU's, capacidade de memória, etc), enquanto a operação *scaling out* cria mais máquinas virtuais. Já quando é necessário diminuir recursos, pode ser feita a ação de *scaling down*, que reduz os recursos internos de uma VM, ou pode-se executar a operação *scaling in*, que reduz o número de VMs atribuídas a uma aplicação. As operações do tipo *scaling out* e *scaling in* são consideradas de redimensionamento horizontal e as ações restantes *scaling up* e *scaling down* como redimensionamento vertical.

Subindo o nível de abstração, ao invés de realizar o dimensionamento de máquinas virtuais, podemos orquestrar contêineres, que executam dentro dessas máquinas ou diretamente no metal. Um contêiner pode ser definido como um ambiente isolado e portável no qual é possível instalar uma aplicação, adicionar bibliotecas, binários e ainda uma configuração básica de como a aplicação deve ser executada [Casalicchio and Perciballi 2017]. Quando executamos uma imagem, que é a descrição de um contêiner, criamos uma instância do mesmo. Podemos criar ou excluir cópias dessas instâncias dependendo da necessidade dos usuários da aplicação em questão. Essas cópias são definidas como réplicas.

Instâncias de um contêiner têm seu tempo de vida administrado por um gerenciador de contêineres [Casalicchio 2019], por exemplo: Docker [Merkel 2014], Apache Mesos [Apache Software Foundation 2018] e Amazon ECS [Amazon Web Services 2018].

Já os orquestradores de contêineres permitem a seleção, implantação, monitoria e controle dinâmico de contêineres em um ambiente de *cloud* [Casalicchio 2019]. Esse tipo de orquestrador deve se preocupar com o controle de recursos em diversos nós, escalonamento de instâncias dentro do *cluster*, balanceamento de carga, checagem de saúde, tolerância a falhas e dimensionamento automático das instâncias ativas. Dessa maneira, os gerenciadores de contêineres atuam em um nível local, enquanto os orquestradores atuam a nível de *cluster*, orquestrando as instâncias entre diversos nós de uma rede. São exemplos desse tipo de orquestrador: Kubernetes [Kubernetes 2018c], Docker Swarm [Docker 2018] e Mesosphere Marathon [Mesosphere 2018]. É importante ressaltar que os orquestradores citados, até o momento da escrita deste artigo, não lidam com orquestração em diferentes domínios administrativos, um dos diferenciais deste trabalho.

Para ilustrar os conceitos apresentados, pode-se pensar em um dos casos de uso deste trabalho, no qual temos uma imagem de um contêiner contendo uma aplicação de vídeo sob demanda, que é instanciada em cada nó do nosso *cluster* pelo Docker e tem suas réplicas gerenciadas globalmente pelo Kubernetes. O Kubernetes possui um nível de organização básico de contêiner chamado *pod*, que representa um processo rodando dentro do *cluster* [Kubernetes 2018b]. Um *pod* encapsula um ou mais contêineres, que vão compartilhar recursos de armazenamento, um IP e outras configurações definidas na sua descrição.

O Kubernetes já conta com um dimensionador automático de *pods* nativo chamado: *Horizontal Pod Autoscaler* [Kubernetes 2018a]. Contudo, problemas em relação às métricas utilizadas por tal dimensionador foram apontados em [Casalicchio and Perciballi 2017]. [Casalicchio and Perciballi 2017] avalia quais são os melhores tipos de métricas para predição da quantidade necessária de *pods* de acordo com a demanda corrente. São definidos dois tipos de métricas: i) métricas relativas, que medem a porção de recursos que cada contêiner utiliza e ii) métricas absolutas, que representam a utilização real de recursos do sistema físico ou VM. Eles propõem um algoritmo que utiliza apenas métricas absolutas. É demonstrado que, para os *benchmarks* testados, as métricas absolutas são melhores na estimativa do número de réplicas necessárias do que as métricas relativas, utilizadas nativamente pelo Kubernetes. Porém, eles lidam apenas com casos de *scale out*, ou seja, aumentando o número de *pods*, não abordando mecanismos para executar *scale in*, que será abordado no decorrer deste trabalho.

Tratando-se da orquestração de *slices*, o trabalho [Sciancalepore et al. 2017] particiona os recursos em duas frentes de gestão: rede e dispositivos IoT. Nesse sentido, cria funções de maximização de uso da rede e dos dispositivos IoT, baseando-se em prioridade. Por se basear em regras estáticas, a abordagem tende a não ser adaptável às variações de demanda, o que não seria aplicável no contexto deste trabalho.

Devido a natureza multidomínio dos recursos que compõem as *slices* no projeto NECOS, nossa proposta é aplicar conceitos osmóticos com a finalidade de manter o equilíbrio entre os recursos das partes da *slice*, migrando recursos excedentes de uma parte pouco utilizada para uma parte que esteja mais sobrecarregada. É importante ressaltar que estamos tratando a migração de recursos como sendo uma deleção do recurso X de uma parte A da *slice* e a criação desse mesmo recurso X em uma parte B. Isso se dá porque se trata de recursos físicos ou virtualizados, que dificilmente poderiam ser migrados literalmente pela *slice*. Pesquisas que já exploraram a Computação Osmótica

são mostradas adiante na Seção 2.1.

2.1. Computação Osmótica

Um novo paradigma que emerge da relação entre as novas estruturas de computação: *IoT*, *Edge Computing* (pequenos *data centers* na borda da rede) e *Cloud Computing* é a computação osmótica, ou do inglês *Osmotic Computing* [Villari et al. 2016]. O objetivo desse modelo de computação é permitir a implantação automática de micro serviços em ambientes altamente distribuídos e federados interconectados entre estruturas de *edge* e *cloud*. Pegando o termo emprestado da biologia, a computação osmótica visa migrar os micro serviços, geralmente na forma de contêineres, entre nuvem e borda de maneira fluída e automática. Essa migração de serviços entre partes da estrutura de um mesmo provedor, ou provedores diferentes, é o que inspira o uso desse paradigma na orquestração de recursos do NECOS, já que o projeto abrange a gerência da *cloud* até a *edge*.

O trabalho [Carnevale et al. 2018] propõe a arquitetura, ainda não implementada, de um orquestrador capaz de implantar micro serviços em um ambiente osmótico, no qual as aplicações podem migrar entre as camadas: *cloud*, *edge* e *IoT*. O trabalho foca na abstração de serviços, que são transformados em "MicroELementoS" (MELS) conteinerizados e na proposta abstrata de um orquestrador. A orquestração é tratada como um problema de otimização multiobjetivo, levando em conta o consumo de energia, o custo e também métricas de disponibilidade. Em tal proposta, os dispositivos IoT e os MELS são cadastrados em um *dashboard* para depois serem orquestrados pelo *Smart Orchestrator*.

Na proposta anterior, o *Smart Orchestrator* possuiria os módulos: gerenciamento de contêineres, gerenciamento de fluxo de dados, módulo de treinamento e módulo de predição. O módulo de fluxo de dados, pré-processaria as métricas enviadas pelos dispositivos, a fim de facilitar as tarefas de treinamento e predição. Já os módulos de treinamento e de predição seriam os responsáveis pelo aprendizado do orquestrador osmótico. O módulo de treinamento utilizaria *Deep Learning* [LeCun et al. 2015] para aprender padrões à partir das métricas enviadas pelos Agentes Osmóticos, a fim de gerar manifestos dinâmicos para a implantação mais assertiva dos MELS. Os modelos seriam salvos em um banco de dados. Já o módulo de predição utilizaria os modelos anteriores de uma implantação para a geração de novos manifestos que poderiam ser mais úteis na implantação dos MELS em diferentes níveis (IoT, *edge*, *fog* e/ou *cloud*).

No nosso trabalho, utilizaremos uma arquitetura similar, porém, tendo um módulo de predição de QoS, para saber a qualidade do serviço que está chegando nos usuários finais, assunto tratado em [Pasquini and Stadler 2017]. Na nossa abordagem, além das métricas da infraestrutura, o QoS recebido pelos usuários finais também será usado no treinamento do modelo de aprendizagem, o que deixará nosso orquestrador ciente da qualidade que chega aos mesmos. Vamos investigar quais algoritmos utilizar no módulo de treinamento para encontrar as causas raiz da violação de QoS e também para detectar o super provisionamento de recursos, assuntos não tratados em [Carnevale et al. 2018].

Em [Sharma et al. 2017] fica claro que o uso da *Fog Computing* pode acarretar em altos custos e redundância de recursos. Essa situação se torna ainda mais complexa quando se trata da administração de serviços heterogêneos, os quais são mais eficientemente executados em dispositivos específicos. Para contornar esse problema, [Sharma et al. 2017] propõe o uso da computação osmótica como forma de balancear os

serviços onde seriam mais eficientemente computados. Para tal, o artigo define uma taxonomia que liga os termos da computação osmótica ao problema a ser resolvido:

Soluto: parte solúvel a qual não é permitida atravessar a membrana semipermeável. No contexto do artigo [Sharma et al. 2017]: poder computacional, energia, tempo de processamento e carga atual.

Solvente: componente que absorve o soluto, podendo se mover pela membrana semipermeável. Neste caso, solvente são serviços que se deslocam entre os servidores.

Solução: composta pela infraestrutura, usuários, servidores, serviços e recursos disponíveis.

Membrana semipermeável: é o elemento responsável por manter a concentração apropriada da solução. No artigo, a camada de servidores na *fog* é responsável por esse papel. Para manter esse equilíbrio é preciso considerar: quando o serviço deve ser movido, para onde deve ser movido (um *data center* ou um dispositivo na *edge*?) e, por fim, como mover esse serviço (usando contêineres ou uma agregação de contêineres?).

Concentração: razão de soluto ou solvente, em computação osmótica pode ser interpretada como a razão do número de serviços a serem oferecidos pelos recursos computacionais disponíveis.

[Sharma et al. 2017] divide os serviços heterogêneos nas categorias micro e macro, com base nos requisitos de recursos computacionais, energéticos e de processamento. Além da taxonomia, o artigo propõe um algoritmo iterativo que visa balancear esses serviços entre as camadas *fog* e *cloud*, até que haja uma distribuição com um certo grau de tolerância ϵ de diferença no balanceamento. Enquanto tal estabilidade não é alcançada, o algoritmo tenta mover os serviços que couberem para a camada osmótica (formando um grupo de micro serviços), e os demais serviços que não couberem nessa camada (grupo de macro serviços) são migrados para a nuvem.

Olhando o contexto da computação osmótica como um meio distribuído de se montar modelos ainda mais inteligentes, [Morshed et al. 2017] exibe os desafios de se aplicar *Deep Learning* para análise de dados clínicos. Sendo o *Deep Learning* uma técnica de aprendizado de máquina que constrói conhecimento a partir de uma hierarquia de conceitos, dos mais concretos aos mais abstratos, ele promove grande flexibilidade e capacidade de aprendizado. No entanto, tal técnica precisa de um grande volume de dados, o que torna seu uso custoso, em termos de transporte e processamento, num contexto de computação distribuída.

Outro desafio do uso desse tipo de aprendizado de máquina quando aliado a computação osmótica, é a heterogeneidade dos dados vindos de diferentes fontes, que em diferentes formatos acrescenta mais complexidade no treinamento de modelos de aprendizagem. Mais um obstáculo seria garantir a privacidade e segurança dos dados, contudo, uma abordagem interessante proposta para evitar esse problema seria compartilhar apenas o resultado do modelo ao invés dos dados "crus" pela rede. Por fim, o trabalho apresenta uma proposta hierárquica de modelagem, na qual os dispositivos na *edge* seriam responsáveis por criar partes de um modelo que é unificado nas camadas superiores da aplicação, sendo que o modelo holístico central ficaria armazenado na nuvem. Tal abordagem de aprendizado pode ser considerada em futuras implementações deste trabalho, considerando as partes da *slice* como a *edge* que computa partes do modelo.

3. Arcabouço para Orquestração Osmótica de *Cloud Slices*

Esta seção utiliza os conceitos apresentados até então para elaborar uma proposta de orquestrador osmótico de *cloud slices* no contexto do projeto NECOS. Apesar do projeto contar com um módulo de orquestração a nível de serviço, o foco deste trabalho se limita na orquestração de recursos da *slice* no modo 0 (infraestrutura). Por esse motivo, neste trabalho, teremos a seguinte analogia osmótica:

Soluto: conjunto de partes de uma *slice*, distribuídas em um ou mais provedores de infraestrutura.

Solvente: conjunto de recursos que compõem uma parte de uma *slice*, que podem ser migrados a fim de balancear a saturação de uma parte da *slice*.

Solução: é composta por todos os elementos de infraestrutura e partes da *slice*.

Membrana semipermeável: representada pelo módulo *Slice Provider* do NECOS, descrito a seguir, será responsável por decidir quais recursos podem ser migrados entre partes da *slice*.

Concentração: razão de carga compartilhada entre todas as partes da *slice*.

A Figura 2 mostra a abstração da orquestração osmótica na conjuntura do projeto. Na base da arquitetura estão os provedores de infraestrutura. Eles podem oferecer recursos de *Data Center* (DC), conexão de *Data Centers* via *Wide Area Network* (WAN), e também outro tipos de recursos (*edge, fog*, etc.) aos *tenants*. Dentro de cada provedor, está um módulo do NECOS chamado *DC Controller*, ou *WAN Controller*, responsável por atuar na infraestrutura do provedor em nome do NECOS.

Um nível acima dos provedores de infraestrutura está o *Slice Provider*, o centro de operações do NECOS, responsável por administrar todas as *slices* solicitadas pelos *tenants*. Dentro dele, existe um módulo de monitoria (*Monitoring Module*), que é encarregado de receber dados de uso e saúde da infraestrutura alocada para as *slices*. Além de receber, ele também pré-processa esses dados, e os redireciona ao *Slice Resource Orchestrator*, que, por sua vez, tomará as decisões de elasticidade necessárias para manter o SLA contratado pelo *tenant*. Mais detalhes sobre a arquitetura do projeto podem ser encontrados em [NECOS 2018a].

No topo da arquitetura, se encontra o domínio dos clientes (*Tenants' Domain*), onde são utilizadas as *slices*. Um cliente nesse caso, são os usuários de *slice* de rede, nuvem ou *data center* no qual serviços customizados são hospedados. As *slices* são compostas por partes. Cada parte é um conjunto de recursos obtidos de apenas um provedor. No projeto NECOS, são definidos dois tipos de elasticidade: horizontal e vertical. Na elasticidade horizontal, são adicionadas ou removidas partes da *slice*, enquanto na vertical, existe o aumento ou a redução de recursos dentro de uma parte da *slice*. O objetivo de se utilizar um orquestrador osmótico é evitar a elasticidade horizontal, usando ao máximo os recursos já alocados, evitando o desperdício de recursos e a complexidade de se contratar mais provedores, possibilitando o aumento de barganha de preços dentro do fornecedor de infraestrutura daquela parte da *slice*.

Dentro do *Slice Resource Orchestrator* haverá um módulo chamado *Osmotic Slice Controller*, responsável por fazer a elasticidade automática das *slices*. Existirá uma instância desse módulo para cada *slice* administrada pelo *Slice Provider*. Dentro do *Osmotic Slice Controller* haverá um laço MAPE de controle customizado, que será implementado, seguindo o fluxo mostrado na Figura 2:

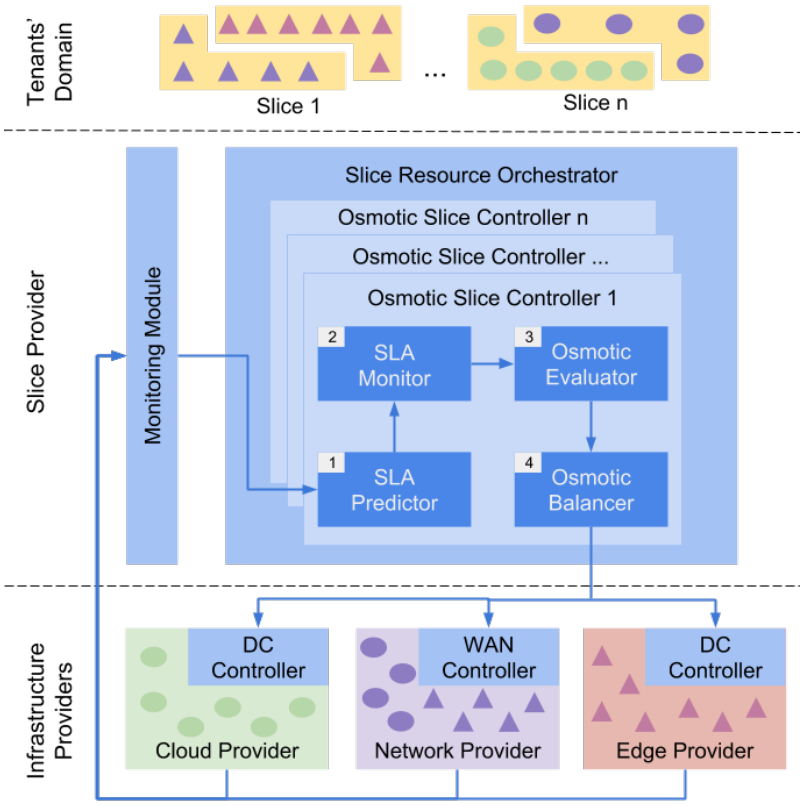

Figura 2. Orquestração osmótica no contexto NECOS. Provedores de infraestrutura (base) fornecem os recursos a serem usados pelos *tenants* (topo), sendo que a gerência é de responsabilidade do *Slice Provider* (meio).

1. O módulo *SLA Predictor* estimará o SLA do usuário final utilizando os dados pré-processados pelo *Monitoring Module*, ou seja, apenas as métricas mais relevantes na estimação do SLA serão levadas em consideração. A predição desse SLA será feita usando algoritmos de aprendizado de máquina do tipo regressão.
2. O fluxo seguirá para o *SLA Monitor*, que será responsável por analisar os SLAs estimados com a finalidade de detectar uma situação ou tendência de degradação da qualidade do serviço, usando algoritmos de reconhecimento de padrão, por exemplo.
3. O *Osmotic Evaluator* analisará a configuração atual da *slice*, para rebalancear os recursos entre as partes da mesma, migrando recursos excedentes de uma parte pouco utilizada para partes que necessitem mais daquele recurso. Nesse ponto, poderão ser usados algoritmos genéticos, para gerar arquivos de implantação que melhor se ajustem à fase corrente do serviço.
4. Por último, o *Osmotic Balancer* executará o plano criado pelo *Osmotic Evaluator*, disparando as ações sugeridas no passo 3 para os receptores do NECOS que residirão dentro dos provedores (*DC/WAN Controller*), usando API's REST.

O módulo *Osmotic Slice Controller* ainda está em fase de concepção. Atualmente, estão sendo testadas algumas hipóteses que validarão a implementação de seus submódulos, como por exemplo:

I - É possível estimar o SLA que o usuário final do serviço está recebendo num

contexto de *cloud slicing*?

II - É possível detectar uma tendência de degradação desse SLA de forma a agir preventivamente na elasticidade dos recursos da *slice*?

III - É possível gerar um arquivo descritivo de partes da *slice* de forma inteligente utilizando-se arquivos descritivos criados anteriormente, juntamente com seus históricos de qualidade de serviço?

Para testar essas hipóteses e o próprio orquestrador depois de implementado, foi implantada a plataforma de experimentação descrita na Seção 4.

4. Plataforma de Experimentação

Para estudo de caso, foi escolhida uma aplicação de vídeo sob demanda que utiliza o recente padrão de *streaming* de vídeos MPEG-DASH [Sodagar 2011]. De acordo com o relatório [Sandvine 2018], quase 58% do tráfego *downstream* de toda internet no ano de 2018 representava *streaming* de vídeos, tornando a avaliação desse tipo de aplicação de extrema importância. O MPEG-DASH é um padrão de *streamming* de vídeo adaptativo, no qual o dispositivo do usuário irá decidir qual segmento, entre os diversos segmentos de vídeo codificado com diferentes qualidades, melhor se adapta a sua largura de banda no momento, visando entregar o vídeo de modo ininterrupto ao usuário.

A plataforma de testes construída para esses experimentos é mostrada na Figura 3. Nela, é possível ver três componentes, seguindo os mesmos padrões de cores da Figura 2: (i) em amarelo, os componentes da *slice*, representando um provedor do tipo *edge*; em azul, representando o módulo de monitoria do NECOS o componente Prometheus; e (iii) um novo componente, até então não apresentado, os usuários da *slice*, que irão consumir o serviço de vídeo disponibilizado na *slice* de um *tenant*. O fluxo dos experimentos ocorre de acordo com a numeração da Figura 3, seguindo os passos:

1. O componente *Kubernetes Master* faz o *deploy* de réplicas do serviço de vídeo dentro do *cluster* de *raspberry pies* e também implanta uma instância do Balanceador de carga dentro de uma máquina virtual (VM), disponibilizada pelo OpenStack [Murphy 2018]. Os contêineres são administrados dentro de cada nó (*raspberry pies* e VM) pelo gerenciador de contêineres Docker [Merkel 2014].
2. A partir do momento que o serviço está em execução, métricas do monitoramento do serviço (contêineres) e das máquinas (virtuais e físicas) são coletadas e mantidas pelo Prometheus [Prometheus 2019]. Esse coletor de métricas executa em uma VM, também disponibilizada pelo OpenStack.
3. Com o serviço pronto para ser consumido, o Gerador de carga é responsável por gerar várias aplicações VLC [VLC 2019], com a finalidade de perturbar a qualidade do serviço oferecido, simulando uma demanda real. Já o cliente modificado é uma versão customizada do VLC, alterada para coletar métricas da qualidade do serviço que está sendo recebido pelo usuário final.

A plataforma foi elaborada de tal maneira que métricas tanto do lado do serviço como do lado dos usuários finais sejam coletadas com a finalidade de se estimar o SLA recebido (Hipótese I), usando algoritmos de aprendizado de máquina supervisionados. A arquitetura também facilita a elasticidade, uma vez que se recursos computacionais precisarem ser adicionados, removidos ou terem suas configurações alteradas, isso pode ser feito virtualmente por meio de chamadas às API's de controle do OpenStack ou do Kubernetes.

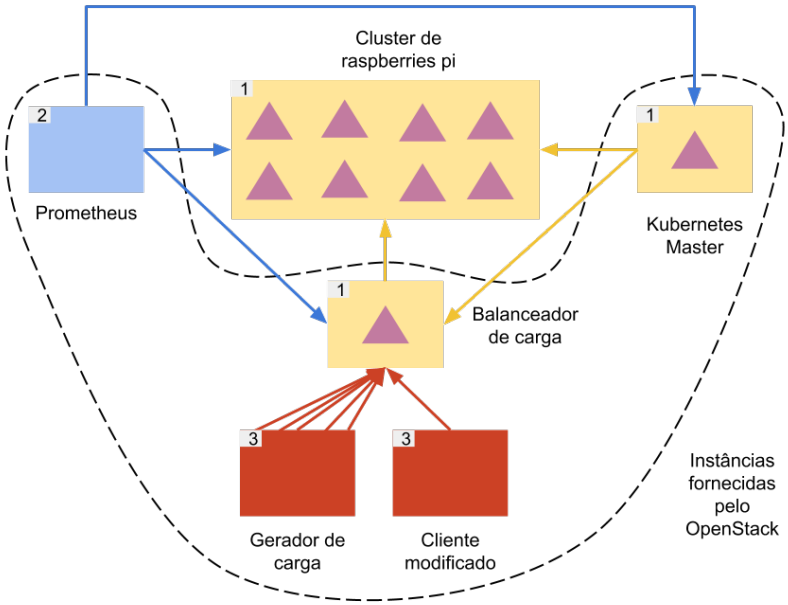

Figura 3. Plataforma de experimentação preliminar

5. Resultados Preliminares

No momento da escrita deste artigo, a plataforma de experimentos encontra-se em fase de calibragem de parâmetros. As variáveis: (i) número de clientes gerados pelo gerador de carga, (ii) número de réplicas do serviço e (iii) largura de banda entre clientes e serviço, precisam estar em uma área na qual seus valores provoquem oscilações realistas na plataforma de experimentos, levando em considerações as limitações da plataforma.

Por esse motivo, experimentos preliminares estão sendo feitos para avaliar, por exemplo, qual a carga máxima de clientes pode ser produzida pelo gerador de carga, qual a largura de banda mínima para os clientes receberem o serviço do servidor, porém, permitindo perturbações na qualidade de serviço. A Figura 4 ilustra um desses experimentos, com as seguintes configurações:

- Limitação de 35 MBits/s na máquina virtual que disponibiliza o serviço de vídeo;
- Carga de clientes com distribuição sinusoidal variando de 1 a 10 clientes, num período de 3600 segundos;
- Lógica de chaveamento de qualidade do tipo *rate* do visualizador de vídeo VLC, o qual utiliza uma estimação da largura de banda da máquina cliente solicitar os próximos trechos de vídeo.

É possível identificar na área entre 500 e 1500 segundos a degradação do vídeo (em azul) com a carga de clientes em alta (em vermelho), enquanto entre 2000 e 3000 segundos, o serviço apresenta estabilidade no recebimento de quadros por segundo, no momento em que a geração de carga está em baixa.

Neste trabalho, o primeiro passo para disparar a orquestração osmótica é a detecção de violações de SLA (Hipótese I). O protótipo construído é capaz de conduzir a carga suportada pelo *slice*, afetando, conforme resultado da Figura 4 o desempenho

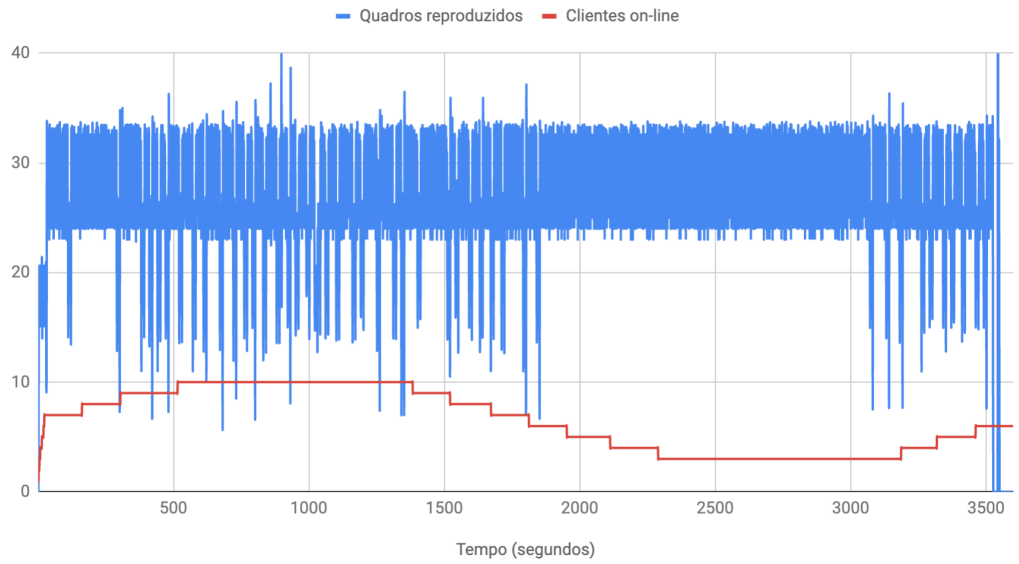

Figura 4. Experimento de uma hora com geração de carga no serviço de *stream* de vídeo.

do serviço em virtude de carga extra sob a infraestrutura. Esta funcionalidade torna automático o disparo da proposta de orquestração osmótica.

6. Conclusão e Trabalhos Futuros

Os estudos teóricos feitos até o momento apontam a computação osmótica como um paradigma promissor na gerência de recursos distribuídos nas três camadas de atuação de aplicações: *cloud, fog* e *edge*, que são o contexto de atuação do projeto NECOS. Além de promissor, o uso da computação osmótica apresenta características favoráveis ao combate ao desperdício dos recursos alocados por diminuir as chances de elasticidade horizontal, o que será investigado adiante nessa pesquisa.

Espera-se como desfecho deste trabalho, a validação de um protótipo de orquestrador de recursos de *slice*, que utilize principalmente o paradigma osmótico para realizar essa tarefa. Dessa maneira, as duas contribuições deste trabalho serão: i) propor um orquestrador inteligente de *slices* sob demanda e ii) expandir os conhecimentos atuais do uso do paradigma osmótico no contexto de gerência de *slices*.

É de nosso conhecimento que o uso da abordagem osmótica acarretará mudanças na camada de serviço, sendo necessária uma sincronização entre as ações do *Slice Resource Orchestrator* e o *Service Orchestrator*, incluindo a migração, criação e deleção de réplicas do serviço nas partes alteradas das *slices*. Essa situação será futuramente analisada, durante a implementação do orquestrador osmótico, avaliando-se o melhor tratamento de cada caso.

Outro trabalho futuro seria analisar casos em que o orquestrador osmótico se mostra mais eficiente do que outras tratativas de redimensionamento de recursos (baseado em regras, regressão, modelos analíticos, etc.). Podendo-se criar então um orquestrador híbrido, que adapta sua estratégia de acordo com exigências do cliente ou conforme a tendência de consumo de recursos apresentada pela *slice* gerenciada.

Agradecimentos

Esta trabalho é financiado com recursos da 4º chamada colaborativa BR-EU no contexto do H2020, registrados no acordo 777067 (*NECOS - Novel Enablers for Cloud Slicing*), que é fomentado pelo Ministério da Ciência e Tecnologia no lado Brasileiro e pela Comissão Europeia de Tecnologia no lado Europeu.

Referências

Amazon Web Services (2018). Amazon elastic container service. `https://aws.amazon.com/pt/ecs/`. Online; acessado em 12-12-2018.

Apache Software Foundation (2018). Apache mesos. `http://mesos.apache.org/`. Online; acessado em 12-12-2018.

Carnevale, L., Celesti, A., Galletta, A., Dustdar, S., and Villari, M. (2018). From the cloud to edge and iot: a smart orchestration architecture for enabling osmotic computing. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 419–424. IEEE.

Casalicchio, E. (2019). Container orchestration: A survey. In *Systems Modeling: Methodologies and Tools*, pages 221–235. Springer.

Casalicchio, E. and Perciballi, V. (2017). Auto-scaling of containers: The impact of relative and absolute metrics. In *Foundations and Applications of Self* Systems (FAS* W), 2017 IEEE 2nd International Workshops on*, pages 207–214. IEEE.

Docker (2018). Swarm mode overview. `https://docs.docker.com/engine/swarm/`. Online; acessado em 12-12-2018.

Freitas, L. A., Braga, V. G., Corrêa, S. L., Mamatas, L., Rothenberg, C. E., Clayman, S., and Cardoso, K. V. (2018). Slicing and allocation of transformable resources for the deployment of multiple virtualized infrastructure managers (vims). In *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 424–432. IEEE.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Kubernetes (2018a). Horizontal pod autoscaler. `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`. Online; acessado em 21-11-2018.

Kubernetes (2018b). Pod overview. `https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/`. Online; acessado em 21-12-2018.

Kubernetes (2018c). Production-grade container orchestration. `https://kubernetes.io/`. Online; acessado em 12-12-2018.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

Mesosphere (2018). Marathon: A container orchestration platform for mesos and dc/os. `https://mesosphere.github.io/marathon/`. Online; acessado em 12-12-2018.

Morshed, A., Jayaraman, P. P., Sellis, T., Georgakopoulos, D., Villari, M., and Ranjan, R. (2017). Deep osmosis: Holistic distributed deep learning in osmotic computing. *IEEE Cloud Computing*, 4(6):22–32.

Murphy, E. (2018). Welcome to openstack documentation. `https://docs.openstack.org`. Online; acessado em 16-03-2019.

NECOS (2018a). D3.1: Necos system architecture and platform specification. v1. `http://www.maps.upc.edu/public/NECOS\%20D3.1\%20final.pdf`. Online; acessado em 12-03-2019.

NECOS (2018b). Novel enablers for cloud slicing. `http://www.h2020-necos.eu/`. Online; acessado em 12-12-2018.

Netto, M. A., Cardonha, C., Cunha, R. L., and Assunçao, M. D. (2014). Evaluating auto-scaling strategies for cloud computing environments. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 187–196. IEEE.

Pasquini, R. and Stadler, R. (2017). Learning end-to-end application qos from openflow switch statistics. In *IEEE Network Softwarization (NetSoft)*, pages 1–9. IEEE.

Prometheus (2019). Overview prometheus. `https://prometheus.io/docs/introduction/overview/`. Online; acessado em 16-03-2019.

Qu, C., Calheiros, R. N., and Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)*, 51(4):73.

Sandvine (2018). The global internet phenomena report. `https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf`. Online; acessado em 18-03-2019.

Sciancalepore, V., Cirillo, F., and Costa-Perez, X. (2017). Slice as a service (slaas) optimal iot slice resources orchestration. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE.

Sharma, V., Srinivasan, K., Jayakody, D. N. K., Rana, O., and Kumar, R. (2017). Managing service-heterogeneity using osmotic computing. *arXiv preprint arXiv:1704.04213*.

Silva, F. S. D., Lemos, M. O., Medeiros, A., Neto, A. V., Pasquini, R., Moura, D., Rothenberg, C., Mamatas, L., Correa, S. L., Cardoso, K. V., et al. (2018). Necos project: Towards lightweight slicing of cloud federated infrastructures. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 406–414. IEEE.

Sodagar, I. (2011). The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67.

Villari, M., Fazio, M., Dustdar, S., Rana, O., and Ranjan, R. (2016). Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83.

VLC (2019). Vlc media player. `https://www.videolan.org/vlc/`. Online; acessado em 16-03-2019.