UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rodrigo Ferreira Martins

Framework para Construção de Aplicações Adaptativas em Nuvem Multimídia

Juiz de Fora

2015

Rodrigo Ferreira Martins

Framework para Construção de Aplicações Adaptativas em Nuvem Multimídia

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

Orientador: Marcelo Ferreira Moreno

Juiz de Fora

2015

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Martins, Rodrigo Ferreira.
Framework para Construção de Aplicações Adaptativas em Nuvem Multimídia / Rodrigo Ferreira Martins. -- 2015.
82 p. : il.

Orientador: Marcelo Ferreira Moreno
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2015.

1. Computação em Nuvem. 2. Multimídia. 3. Adaptabilidade. 4. Descoberta de Recursos. I. Moreno, Marcelo Ferreira, orient. II. Título.

Rodrigo Ferreira Martins

Framework para Construção de Aplicações Adaptativas em Nuvem Multimídia

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

Aprovada em 10 de Setembro de 2015.

BANCA EXAMINADORA

Prof. D.Sc. Marcelo Ferreira Moreno - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Eduardo Barrére
Universidade Federal de Juiz de Fora

Prof. D.Sc. Esteban Walter Gonzalez Clua
Universidade Federal Fluminense

RESUMO

Atualmente os tradicionais computadores de mesa e *laptops* estão perdendo mercado para dispositivos mais pervasivos, como *smartphones*, *tablets* e *wearables* que em sua grande maioria têm limitações de *hardware* devido às restrições de tamanho, peso e duração da bateria. As aplicações mais populares nos dias de hoje envolvem multimídia e algumas vezes consomem mais recursos do que esses dispositivos são capazes de suportar, por exemplo, realidade aumentada, jogos e o uso de computação para estender a capacidade cognitiva como reconhecimento facial e de fala, processamento de linguagem natural, aprendizagem de máquina, planejamento e tomada de decisão. Nesse contexto, mesmo a já tão popular *cloud computing* não serve como solução por si só, uma vez que a latência e o *jitter* criam uma restrição para aplicações interativas. Este trabalho propõe um *framework* para a construção de aplicações multimídia adaptativas que, no lado cliente, permite explorar recursos dos dispositivos alcançáveis, sejam móveis ou não, a fim de tornar as aplicações mais imersivas. Não apenas os recursos quantitativos, mas também os qualitativos são levados em consideração para distribuir as tarefas. Quanto à nuvem, a proposta apropria-se da ideia de *edge cloud computing* para aumentar a Qualidade de Serviço (QoS) e permitir que regras de negócio também sejam levadas em consideração durante a distribuição das tarefas, bem como na sintonização do serviço.

Palavras-chave: Computação em Nuvem. Multimídia. Adaptabilidade. Descoberta de Recursos.

ABSTRACT

Currently traditional desktop computers and laptops are losing market share to more pervasive devices such as smartphones, tablets and wearables that usually have hardware limitations due to restrictions on size, weight, and battery life. The most popular applications today involve multimedia and sometimes consume more resources than these devices are capable of supporting, for example, augmented reality, games, and the use of computing to extend the cognitive ability as facial and speech recognition, natural language processing, machine learning, planning and decision making. In this context, even the widespread cloud computing does not serve as a single-handed solution, since the latency and jitter create a restriction for interactive applications. This work proposes a framework for building adaptive multimedia applications, which, on the client side, allows for exploring resources from reachable devices, whether mobile or not, in order to make applications more immersive. Not only quantitative, but also qualitative resources are considered to distribute tasks. Regarding the cloud side, the proposal appropriates the idea of edge cloud computing to increase Quality of Service (QoS) and allow business rules to be also taken into account during the distribution of tasks and service tuning.

Keywords: Cloud Computing. Multimedia. Adaptability. Resource Discovery.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ADC Application Delivery Controller

API Application Programming Interface

BPaaS Business Process-as-a-Service

BPM Business Process Management

BPMS Business Process Management Systems

CAPEX Capital Expenditure

CDN Content Delivery Network

Codec Coder-Decoder

CPU Central Processing Unit

CSP Cloud Service Provider

DDoS Distributed Denial of Service

DiffServ Differentiated Services

DLL Dynamic-Link Library

DSA Dynamic Site Acceleration

ENIAC Electronic Numerical Integrator And Computer

FEO Front-End Optimization

FLOPS Floating-point Operations Per Second

FPS Frames Per Second

GaaS Game-as-a-Service

GNU GNU's Not Unix!

GPU Graphics Processing Unit

HPC High-Performance Computing

IaaS Infrastructure-as-a-Service

IBM International Business Machines Corporation

IDaaS Identity-as-a-Service

ID Identifier

ILLIAC Illinois Automatic Computer

IntServ Integrated Services

I/O Input/Output

IP Internet Protocol

ISP Internet Service Provider

LISP Locator/ID Separation Protocol

MaaS Management-as-a-Service

MEC Media-Edge Cloud

MFA Multi-Factor Authetication

MIPS Millions of Instructions Per Second

MSP Multimedia Service Provider

mutex mutual exclusion

NFC Near Field Communication

NFV Network Functions Virtualization

NIST National Institute of Standards and Technology

NSP Network Service Provider

OPEX Operational Expenditure

ORDVAC Ordnance Discrete Variable Automatic Computer

OSGi Open Services Gateway initiative

P2P Peer-to-Peer

PaaS Platform-as-a-Service

PC Personal Computer

QaaS Quality-as-a-Service

QoE Quality of Experience

QoS Quality of Service

RAM Random-Access Memory

RSVP Resource Reservation Protocol

SaaS Software-as-a-Service

SDN Software-Defined Networking

SLA Service-Level Agreement

SOA Service-Oriented Architecture

SPI Service, Plaform and Infrastructure

SP Service Provider

SSL Secure Sockets Layer

SSO Single Sign-On

TCP Transmission Control Protocol

TI Tecnologia da Informação

UDP User Datagram Protocol

VPN Virtual Private Network

WSAN Wireless Sensors and Actuators Network

WWW World Wide Web

SUMÁRIO

1 INTRODUÇÃO

Atualmente os tradicionais computadores de mesa e *laptops* estão perdendo mercado para outros dispositivos, como *smartphones* e *tablets*. Os dispositivos móveis não são os únicos dispositivos a se popularizarem recentemente, muitos outros fazem parte do dia a dia das pessoas, por exemplo, consoles de jogos, *smart TV* e *smart wearables* (*watch*, *wristband* e *glasses*).

Alguns desses dispositivos têm em comum o fato de estarem sempre conectados, serem leves e, consequentemente, possuírem limitações nítidas de *hardware* e uso de energia se comparados com os computadores tradicionais.

Outro ponto importante é que com o acesso à Internet aumentando cada vez mais, com maior largura de banda e qualidade, a *World Wide Web* (WWW) deixou de ser uma plataforma majoritariamente dedicada ao texto para incluir multimídia (e.g. vídeo, áudio e imagens). Existem ainda outras aplicações populares usando a Internet que não são baseadas na *Web*, como aplicações de visualização remota, jogos e videoconferências, que além dos tradicionais requisitos de processamento e armazenamento, precisam obter garantias de tempo (e.g. retardo máximo, variação estatística do retardo e sincronização), para uma boa experiência devido à sua natureza conversacional interativa.

Como solução para dispositivo leves, muito se fala de computação em nuvem, para tirar o fardo de processamento, armazenamento e consumo de energia dos mesmos. A computação em nuvem é um modelo que proporciona às aplicações acesso a um conjunto de recursos computacionais compartilhados sob demanda, de forma flexível, conveniente e ubíqua, no qual tais recursos podem ser rapidamente alocados e liberados sem que se exija grande esforço de gerenciamento ou intermediação do provedor do serviço de nuvem (MELL; GRANCE, 2011).

No entanto, alguns problemas surgem dessa abordagem, como a latência, principalmente em aplicações interativas, além de ociosidade ou mesmo desperdício de recursos do dispositivo cliente, que são de uso mais vantajoso dependendo do contexto e requisitos da aplicação.

Arquiteturas de borda de nuvem baseadas no conceito de *fog computing*[1] (BONOMI et

[1]Fog significa nevoeiro que é uma nuvem próxima do chão.

al., 2012), *cloudlet*[2] (SATYANARAYANAN et al., 2009) e *cyber foraging*[3] (BALAN et al., 2002; SHARIFI et al., 2012) vêm sendo propostas (ZHU et al., 2011; ISLAM; GRéGOIRE, 2012).

Tais tipos de abordagens "híbridas" são importantes, pois as nuvens de propósito geral não são projetadas para atender requisitos específicos de aplicações multimídia, como atraso e *jitter*[4]. Quanto aos problemas de ociosidade e desperdício, esses acontecem, pois é mais fácil desenvolver uma aplicação que execute inteiramente em um ambiente controlado, seja no cliente ou na nuvem.

Nesse contexto, o presente trabalho propõe um *framework* para a construção de aplicações multimídia adaptativas, no qual tanto os recursos da nuvem quanto os localizados próximos ao cliente sejam levados em consideração para atingir a melhor experiência possível com a aplicação. Recursos, nesse caso, vão além de capacidade de processamento e armazenamento: em resumo, os recursos podem ser quantitativos ou qualitativos, de *hardware* ou de *software*. Além disso, para a tomada de decisão na alocação e realocação de recursos envolvendo nuvem e cliente, outros fatores são levados em consideração, como contexto da aplicação e regras de negócio.

Para promover a utilização de tal *framework*, propõe-se que suas funcionalidades sejam simples de usar do ponto de vista do desenvolvedor de aplicações, oferecendo um *middleware* que facilite a criação de aplicações distribuídas, pervasivas e adaptáveis ao tornar transparente a heterogeneidade existente nesse cenário (SANAEI et al., 2014).

Ao envolver cliente e nuvem na negociação e sintonização do serviço, espera-se atender a requisitos avançados, como a reconfiguração dinâmica na presença de anomalias, adaptação de conteúdo, provisionamento de qualidade de serviço (*Quality of Service* — QoS) como um serviço e *green computing* (BALIGA et al., 2011), dado que o *middleware* suporta políticas dependentes de contexto, que o habilitam como um *energy-aware middleware* (PETRE, 2008).

Do ponto de vista arquitetural, o *framework* apresenta os componentes necessários para o suporte a aplicações adaptativas cientes de nuvem multimídia, ao apropriar-se e aperfeiçoar os conceitos de *Media-Edge Cloud* (MEC) e *Multimedia Service Provider*

[2]Cloudlet significa pequena nuvem.

[3]Foraging (forrageamento) deriva da ecologia comportamental que significa a busca e a exploração de recursos alimentares. Alguns modelos matemáticos utilizam o ganho entre o que se obtém de energia e o que se gasta ao buscar determinado item alimentar.

[4]Jitter é a variação estatística do atraso na entrega de dados.

(MSP) propostos em Zhu et al. (2011). O MEC é o componente de borda de nuvem fornecido pelo *Cloud Service Provider* (CSP) e independente do MSP. Em tal cadeia de valor, um CSP pode oferecer os recursos de seus MECs para diferentes MSPs, bem como um MSP pode fazer uso de vários MECs de um ou mais CSP.

Resumidamente, quando o cliente inicia o uso de uma aplicação que explora o *framework* proposto, essa entra em contato com o MSP que o direciona para o MEC mais próximo[5] a fim de obter uma melhor qualidade de experiência (*Quality of Expecience* — QoE). Aplicações multiusuário podem acontecer entre MECs desde que exista um acordo entre eles para garantir QoS. Aplicações sob o domínio de um único MEC são mais fáceis de se garantir QoS, pois a infraestrutura normalmente possui um único responsável e uma política uniforme, não precisando de acordo com terceiros.

Esta dissertação está organizada da seguinte forma: O Capítulo chap:theoreticalBackground apresenta um breve histórico sobre a evolução da computação distribuída até chegar à computação em nuvem, seguido de um levantamento teórico dessa última. O Capítulo 3 discute trabalhos relacionados a aspectos como suporte multimídia em nuvem, distribuição de tarefas, QoS e borda da nuvem. O Capítulo 4 aponta e descreve os principais componentes da arquitetura do ponto de vista de uma cadeia de valor, seus atores e *workflow* de provisinamento de um serviço em cima dessa estrutura. O Capítulo 5 apresenta os principais componentes do ponto de vista técnico detalhando de seus sub-componentes e funcionamento. O Capítulo 7 apresenta questões sobre o desenvolvimento do *middleware* e mostra o seu atual estado, bem como uma aplicação usando sua API onde ambos servem como prova de conceito. O Capítulo 8 trás considerações adicionais para o trabalho proposto, como questões de latência, segurança e ecossistema. O Capítulo 9 mostra os principais pontos de pesquisas que ainda devem ser feitas sobre o *framework*. E por fim o Capítulo 10 trás as conclusões.

[5]O conceito de próximo é definido pelas políticas e regras de negócio estabelecidas pelo MSP. Nem sempre será escolhido o MEC mais próximo em termos de localização, número de saltos ou menor retardo.

2 ARCABOUÇO TEÓRICO

Para a compreensão da arquitetura proposta e suas vantagens deve-se primeiramente compreender o conceito de computação em nuvem (*cloud computing*) (RIMAL et al., 2009), suas vantagens e desvantagens e como tirar proveito dela. A origem do termo não é clara, havendo indícios da primeira aparição no ano de 1996 (REGALADO, 2011), mas não existe nenhum pedido de marca registrada antes de 1997 nos Estados Unidos (CORPORATION, 1998). Quanto à sua definição, existem muitas. Para a pergunta "O que é *cloud*?" alguns dirão "É um *cluster*!", "É um supercomputador!", "É um *datastore*!", "Não, é o *superman*!", outros dirão que nenhuma dessas alternativas ou todas elas. A definição em que se baseia o presente trabalho é a do NIST (MELL; GRANCE, 2011):

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

Nessa definição está incluído o Modelo SPI (*Service, Plaform and Infrastructure*), conforme Figura 2.1, e os cenários de implantação, nuvem privado, nuvem comunitário, nuvem pública e nuvem híbrida.

2.1 HISTÓRIA

Hoje em dia computação em nuvem é um jargão da informática muito popular, mas esse não é de forma alguma o primeiro sistema de computação distribuída. Os primeiros computadores construídos com arquitetura similar à usada nos dias de hoje datam da década de 1940, por exemplo, ENIAC, ORDVAC e ILLIAC. Eles eram verdadeiras centrais de dados (*data centers*) que ocupavam galpões e prevaleceram até o fim da década de 1950.

As décadas de 1960 e 1970 compreenderam a era das companhias de *time-sharing* e indústria de processamento de dados. Também nessas décadas a IBM fez muitos progressos com a virtualização.

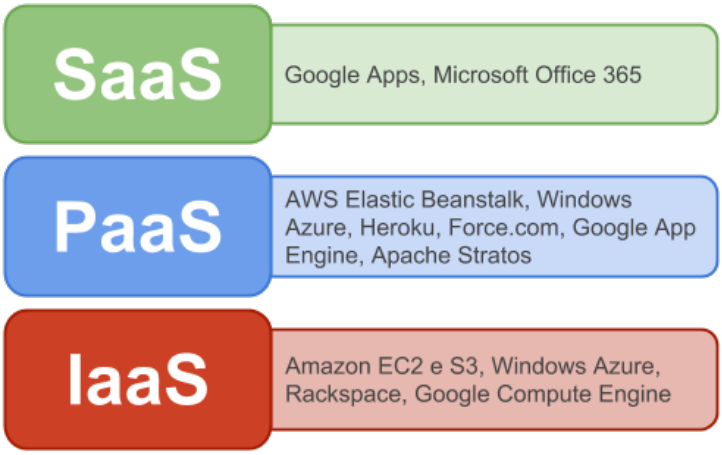

Figura 2.1: Modelo SPI: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS)

Na década de 1980 os computadores pessoais (*Personal Computer* — PC) se popularizaram tornando mais fácil a construção de *clusters*, em detrimento das companhias de *time-sharing* e indústria de processamento de dados. Os PCs também levaram a outras evoluções na computação, tais como computação em grade (*grid computing*), e sistemas de grande escala como *Peer-to-Peer* (P2P), que surgiu na década de 1990.

Aconteceram outros dois eventos importantes na década de 1990 para completar esse cenário pré-nuvem. Primeiro, as companhias de telecomunicações começaram a oferecer serviços de *Virtual Private Network* (VPN) com tão boa QoS e menor custo do que os até então oferecidos *links* ponto-a-ponto. Segundo, no dia 7 de agosto de 1991 a WWW tornou-se publicamente disponível, o que impulsionou a Internet como um todo.

Como a computação se desenvolve em um grande *loop*, pode-se considerar que, atualmente, essa retornou a era das indústrias de *time-sharing* e processamento de dados. Diferente das décadas de 1960 e 1970 que usavam *mainframes*, atualmente usa-se *clusters* e computação de alto desempenho (*High-Performance Computing* — HPC) para tratar grandes quantidades de dados. Inclusive, a escala de processamento muito maior é um dos diferenciais modernos, junto às diferentes cargas de trabalho atuais. Devido à maior escala, é comum surgir o termo computação paralela que se divide em duas abordagens para atender dois tipos de aplicações: aplicações *CPU-bound* (*compute-intensive*) e aplicações *I/O-bound* (*data-intensive*). Alguns trabalhos sugerem dimensões adicionais para a classificação da aplicação, como *Paging-bound* (*memory-intensive*) e *network-intensive*. Esse conceitos são importantes, pois conhecendo a natureza da aplicação aumenta-se a

qualidade do provisionamento de recursos (ZHANG; FIGUEIREDO, 2006).

Como foi visto, computação distribuída não é um conceito novo e até mesmo a ideia por trás da computação em nuvem já havia sido pensada muito antes de ser concebida. John McCarthy no evento MIT Centennial em 1961 (GARFINKEL; ABELSON, 1999) disse publicamente que o *time-sharing* poderia levar a um futuro onde a tecnologia da computação seria vendida seguindo um modelo de negócio baseado em *utility*, como a água ou a eletricidade.

> If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility [...] Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system [...] Certain subscribers might offer service to other subscribers [...] The computer utility could become the basis of a new and important industry. (John McCarthy, 1961)

Hoje, já se observa a computação como a quinta utilidade depois da água, eletricidade, gás e telefonia (BUYYA et al., 2009). No atual modelo, recursos compartilhados como processamento e armazenamento são oferecidos sobre a Internet segundo a ideia de *utility computing*. A capacidade de autogerenciamento desses recursos distribuídos e adaptação a mudanças imprevisíveis é chamado de *autonomic computing*. Esse conceito se estende para os sistemas auto* (*self-**) (BABAOGLU; JELASITY, 2008) que indicam variados modos de adaptação automática dos serviços, como, autogerenciamento, autorreparação, autoconfiguração, auto-otimização, etc. Ver Figura 2.4 no final do capítulo.

2.2 VANTAGENS DO ARMAZENAMENTO EM NUVEM

Dentre as vantagens de se usar a computação em nuvem está o armazenamento "ilimitado", onde os usuários podem aumentar sua demanda por espaço indefinidamente sem o risco de não ter onde colocar seus arquivos. Problemas físicos/lógicos com o mesmo que geram a necessidade por *backups*, são um fardo que ao usar armazenamento em nuvem pertencem ao provedor desse serviço.

O uso do armazenamento em nuvem traz características que são impossíveis ou muito limitadas quando usando armazenamento local, como a possibilidade de acesso ubíquo aos

dados independente de hora e lugar.

Outro recurso interessante é a capacidade de compartilhar arquivos sem a preocupação de não poder desligar o computador enquanto outro estiver o acessando, tornando o compartilhamento assíncrono. Ainda pensando no compartilhamento, é fácil chegar à conclusão que o acesso cliente-servidor é mais rápido do que o acesso cliente-cliente, em se tratando de usuários geograficamente distantes. Sem contar que o acesso cliente-cliente não é *firewall-friendly*. É mais simples também o compartilhamento um para muitos, onde o portador do conteúdo faz o *upload* apenas uma vez e várias pessoas podem baixar sem ocupar sua rede.

2.3 VANTAGENS DO PROCESSAMENTO EM NUVEM

Principalmente para dispositivos móveis com poucos recursos, esse modelo é muito interessante, não só devido ao armazenamento, mas também ao processamento limitado.

Algumas aplicações não são factíveis em dispositivos mais leves devido as limitações de recursos, assim a computação em nuvem cria novas possibilidades. Outro agravante nesses dispositivos é o uso da bateria, que também se torna um benefício para o uso do processamento externo.

Mesmo quando falamos de corporações de pequeno ou grande porte, a computação em nuvem se mostra interessante, pois permite escalabilidade e disponibilidade a um baixo custo. A escalabilidade diz respeito a capacidade de aumentar os recursos quando a demanda aumenta, mas esse conceito se estende para a elasticidade, onde não só os recursos podem crescer para atender a demanda, como também podem diminuir para fins de economia.

Quando empresas mantêm seus próprios servidores, elas devem dimensioná-los para suportar os picos de carga e no resto do tempo o *hardware* fica ocioso, o que representa prejuízo para as empresas. Por tanto as empresas se beneficiam da nuvem por se moverem do modelo *Capital Expenditure* (CAPEX) para o modelo *Operational Expenditure* (OPEX), onde vão pagar por um serviço em vez de um ativo (TAURION, 2010). Ainda em termos financeiros, o custo com a mão de obra para manter servidores, o local e a energia, muitas vezes é maior do que o custo com o serviço em nuvem.

Como a cobrança em IaaS é baseada no uso (*pay-as-you-go*), quanto mais eficiente for a capacidade do sistema em se adequar à demanda, menor será o gasto desnecessário.

Muitas pesquisas são feitas no que se refere a elasticidade.

2.4 VANTAGENS DO SOFTWARE EM NUVEM

Não só a parte de infraestrutura é vendida como um serviço, mas também *softwares* e plataformas podem usar desse mesmo modelo. Uma das vantagens proporcionada por esse tipo de serviço é transferência de responsabilidades para terceiros, como, instalação, manutenção e atualização dos *softwares*, permitindo que o usuário se foque nas tarefas que realmente o interessam.

Outra vantagem é a redução de custos, pois alguns *softwares* são caros e muitas vezes inacessíveis para o usuário final ou mesmo empresas. Com o pagamento por demanda os custos reduzem permitindo que usuários tenham acesso a mais tecnologia e empresas se tornem mais competitivas.

Com o *software* em nuvem fica fácil a integração com outros serviços, como compartilhamento e armazenamento.

Um exemplo dessas vantagens para os desenvolvedores ocorre quando mudam de computador ou começam um novo projeto. Eles precisam configurar todo o ambiente de desenvolvimento além de transferir arquivos do projeto para a nova máquina. Essas tarefas não são necessárias usando PaaS. Esse tipo de serviço permite também a integração com outros serviços, por exemplo, testes automatizados.

Usuários domésticos frequentemente precisam formatar suas máquinas ou atualizar *softwares*, o que gera muitos transtornos devido ao desconhecimento técnico para executar a tarefa, ou pelo trabalho de realizar *backups*. Todas essas tarefas se tornam muito simples ou irrelevantes ao usar SaaS.

2.5 DESVANTAGEM DE SERVIÇOS EM NUVEM

Uma grande desvantagem está na complexidade de se criar aplicações para funcionar em nuvem, pois elas normalmente envolvem conceitos e técnicas mais avançadas como processamento em paralelo, sincronismo e afins.

Devido a falta de padrões, a transparência em QoS e gerenciamento são questionáveis. Segurança de dados, processos e aplicações é outro problema recorrente devido à imaturidade das políticas e normas de segurança em nuvem. Como ter certeza se os dados

estão realmente seguros? Como ter certeza que o sigilo das informações será mantido? A empresa é realmente capaz de cumprir o *Service-Level Agreement* (SLA) e garantir a disponibilidade e QoS? Esses e outros questionamentos são feitos (BADGER et al., 2012).

A heterogeneidade também é um fator complicador, seja ela dos serviços, da QoS desses serviços, da rede ou dos dispositivos.

Essas desvantagens apenas acarretam uma sobrecarga para as soluções, mas de maneira alguma inibem o uso da nuvem, pelo contrário, elas abrem espaço para mais pesquisas em prol de soluções cada vez mais simples e estáveis. Por esses motivos, hoje a computação em nuvem já é realidade, com muitas soluções consideravelmente seguras, adaptáveis à heterogeneidade e mesmo sendo complexas são ainda viáveis.

2.6 CLOUD 2.0

Atualmente as tecnologias de virtualização e de *cluster*[6] são a base da computação em nuvem. A nuvem também se embasa nos conceitos provenientes da *utility*, *autonomic computing* e da *Service-Oriented Architecture* (SOA). As normas e boas práticas dessa arquitetura permitem que os recursos sejam disponibilizados de forma padronizada e simples.

Enquanto antigamente todas as tarefas da Tecnologia da Informação (TI) eram de responsabilidade da empresa, com o passar do tempo elas estão delegando tarefas com menor valor de negócio para terceiros, no caso, a nuvem. Isso permite às empresas se concentrarem no núcleo de suas atividades. Isso é particularmente interessante para *startups*, que adquirem competitividade. Ver Figura 2.2.

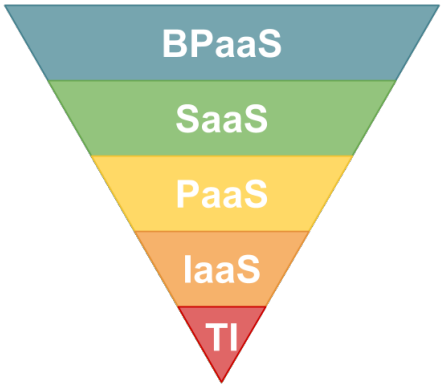

Figura 2.2: Pirâmide com representação do valor de negócio (ASHAR, 2013)

Os populares IaaS, PaaS, SaaS têm sido estendidos para tornar mais clara as diferentes tarefas envolvidas em um negócio e permitir que as empresas se foquem cada vez mais no seu diferencial.

O *Management-as-a-Service* (MaaS) fornece um serviço transversal de gerenciamento

[6]O termo cluster deriva da biologia e significa grupo de similares.

de políticas, segurança, autenticação, recuperação de desastres, cobrança, provisionamento, planejamento de capacidade, monitoramento e gerenciamento de sistemas.

O *Business Process-as-a-Service* (BPaaS) junta conceitos de *Business Process Management* (BPM) com aspectos da nuvem. Os *Business Process Management Systems* (BPMS) integram os processos de negócio. Um BPMS coordena a execução dos processos e fornece informações de progresso dos mesmos. Com BPaaS os processos de negócio são entregues como um serviço permitindo o uso de todas as vantagens da nuvem. Serviços empresariais (oferecidos como SaaS) serão orquestrados (como BPaaS), gerenciado e monitorado (como MaaS), executado (como PaaS) e hospedado (como IaaS) inteiramente em nuvem (ASHAR, 2013). Ver Figura 2.5 no final do capítulo.

Além do foco no núcleo do negócio, outros tópicos estão se popularizando e construindo a nova visão da nuvem. Alguns temas em alta, são a computação móvel e as mídias sociais. Uma evolução natural em computação é a adoção de padrões abertos para permitir intercomunicação, por exemplo, para formar as nuvens híbridas. Ver Figura 2.3.

Figura 2.3: Características da nuvem durante sua evolução (ASHAR, 2013)

Por último, para construir o cenário da *Cloud 2.0*, melhorias nos serviços existentes são necessárias, como as que seguem abaixo e são apresentadas por Johnson (2013) para a IaaS.

- **Faturamento por minuto:** granularidade mais fina na cobrança;
- **Flexibilidade:** permitir a escolha independente de cada item do *hardware*, por exemplo, o número de núcleos de *Central Processing Unit* (CPU) ou *Graphics Processing Unit* (GPU), quantidade de *Random-Access Memory* (RAM) e capacidade de armazenamento;
- **Escalabilidade vertical (*scale-up*):** aumentar os recursos de uma máquina, em vez de aumentar o número de máquinas (escalabilidade horizontal ou *scale-out*). Mais adequado do ponto de vista das aplicações;

- **Desempenho consistente:** o superdimensionamento faz com que máquinas com a mesma configuração tenham um *benchmark* diferente. Para solucionar isso é possível fornecer os recursos a partir de um *pool*, bem como usar melhores técnicas de virtualização;

- **Facilidade de uso:** montar a arquitetura usando ferramentas gráficas.

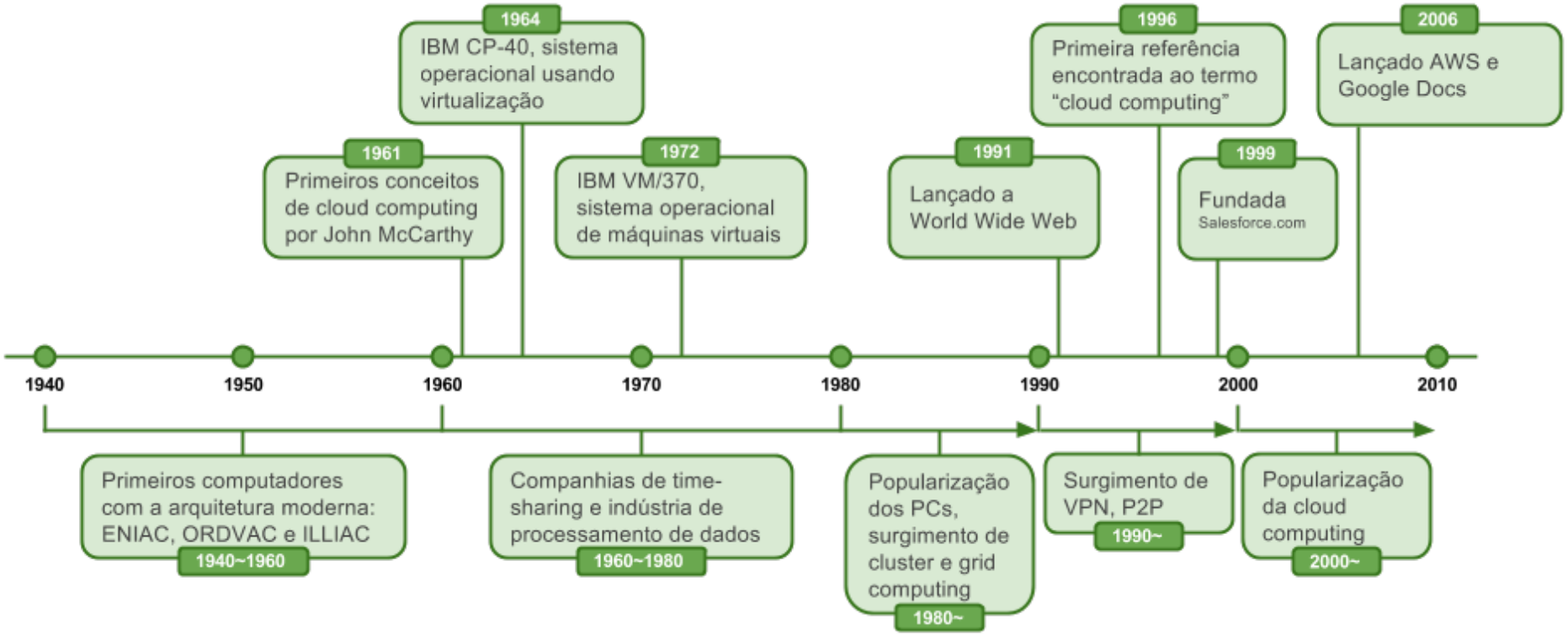

Figura 2.4: Linha do tempo com os principais acontecimentos relacionados a evolução da computação distribuída e computação em nuvem

Figura 2.5: Tabela com responsabilidade de gerenciamento para cada tipo de serviço em nuvem (ASHAR, 2013)

3 TRABALHOS RELACIONADOS

A proposta do presente trabalho se apropria de conceitos e termos definidos em Zhu et al. (2011). Aquele trabalho define conceitos relacionados a *multimedia cloud computing*, a partir de duas perspectivas: nuvem ciente de multimídia (*multimedia-aware cloud* ou *media cloud*) e multimídia ciente de nuvem (*cloud-aware multimedia* ou *cloud media*).

A primeira se refere a como a nuvem pode executar processamento e armazenamento de multimídia distribuídos e prover provisionamento de QoS para esse tipo de serviço. Para atingir alta QoS, os autores propõem uma *cloudlet* denominado MEC. A segunda se refere a como serviços e aplicações multimídia podem tirar proveito dos recursos da nuvem para atingir alta QoE.

A arquitetura definida pelo *framework* na presente proposta utiliza de alguns termos e conceitos cunhados naquele trabalho, como MEC e MSP e estende a pesquisa para definir como uma aplicação é dividida e negociada entre nuvem e cliente.

Em Zhu et al. (2011) é proposto o balanceamento da tarefa em dois níveis. Em nível de usuário (Figura 3.1), todas as tarefas de um usuário são alocadas dentro de um mesmo nó virtual no cluster e os usuários são então distribuídos através dos nós existentes. Em nível de tarefas (Figura 3.2), as tarefas são processadas por vários nós virtuais simultaneamente.

A arquitetura proposta no presente trabalho define que tal distribuição seja estendida, de forma que tarefas diferentes possam ser distribuídas em nós virtuais diferentes ou conjuntos de nós (Figura 3.3). Isso permite que as tarefas sejam alocadas de acordo com os recursos específicos que elas necessitam.

Um trabalho muito próximo à presente proposta é o *framework* AIOLOS (BOHEZ et al., 2014). AIOLOS é um *framework* que visa reduzir a complexidade para o desenvolvedor ao criar aplicações distribuídas e multiusuários para dispositivos móveis em um ambiente heterogêneo. Sua arquitetura contempla também um elemento na borda da nuvem para aumentar a qualidade de serviço e os *data centers* no núcleo da rede. AIOLOS é um *framework* desenvolvido em cima do *Open Services Gateway initiative* (OSGi) e baseado em componentes, os quais podem ser distribuídos entre a nuvem e o cliente.

O trabalho aqui apresentado se diferencia principalmente em três pontos, o primeiro é não ter como objetivo central suprir as limitações de recursos de dispositivos móveis. O

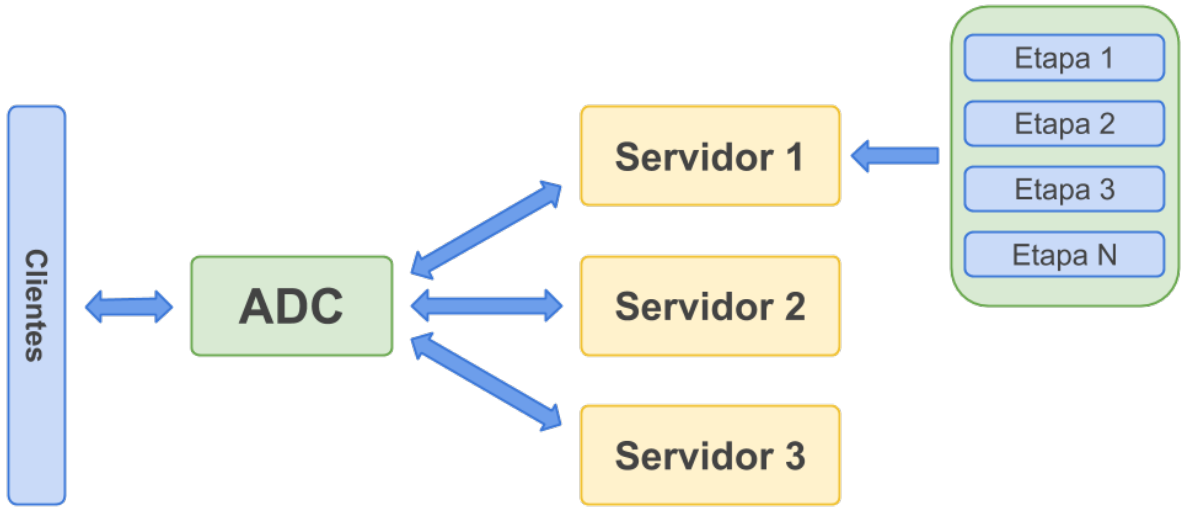

Figura 3.1: Balanceamento de carga em nível de usuário (ZHU et al., 2011)

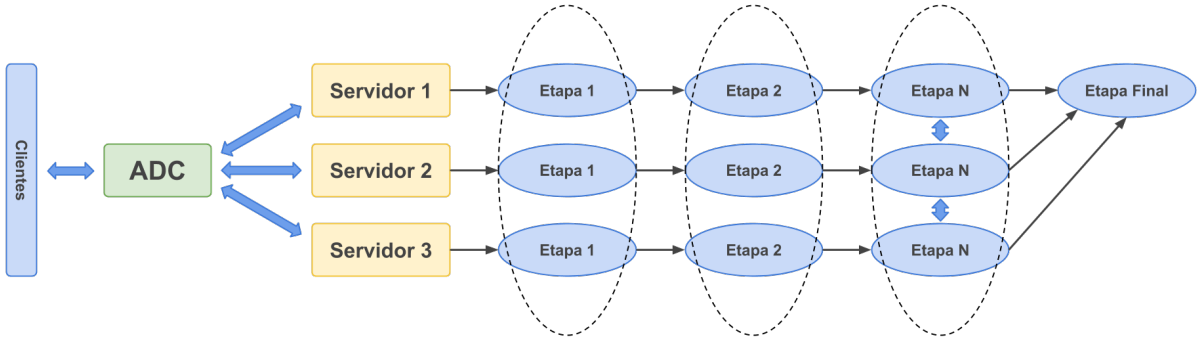

Figura 3.2: Balanceamento de carga em nível de processamento (ZHU et al., 2011)

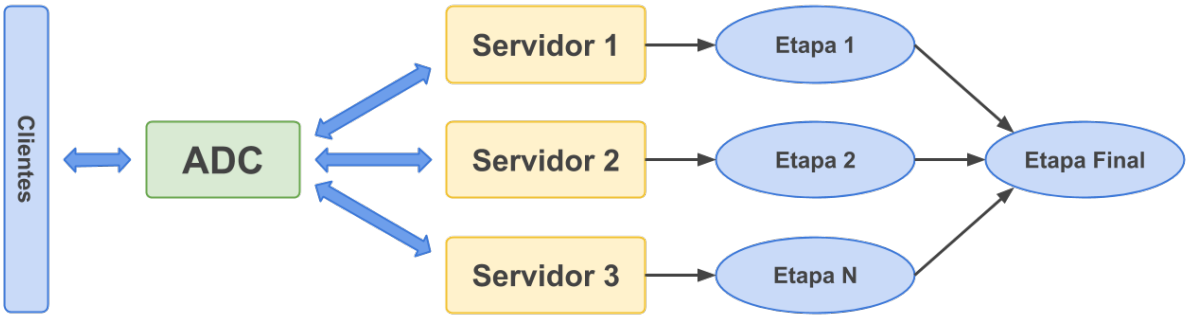

Figura 3.3: Balanceamento de carga em nível de tarefa

trabalho proposto tenta tirar proveito do que há de melhor em cada dispositivo, seja ele móvel ou não, por exemplo, um dispositivo móvel pode precisar de mais recursos de processamento e um computador de mesa pode necessitar de algum sensor para determinado

tipo de aplicação, nesse caso eles podem cooperar na execução. A nuvem também pode fornecer os recursos necessários, como um *cluster* de processamento ou uma rede de sensores. Além disso, o *middleware* aqui proposto leva em consideração recursos quantitativos e qualitativos, permitindo criar políticas de migração mais abrangentes.

A segunda diferença está na inclusão de regras de negócio na política de migração, além dos requisitos de *hardware* e *software*, contexto dos recursos ou análise do funcionamento da aplicação. Isso é particularmente interessante para a definição de uma cadeia de valor aberta, que permita a inclusão de diferentes atores, em diferentes papéis como MSP e CSP, além da possibilidade de criação de diferentes planos de usuários e modelos de serviço (e.g. auxílio computacional ou SaaS). A última grande diferença em relação à Bohez et al. (2014) está na proposta de venda de QoS como um serviço que pode ser feita, por exemplo, usando a diferenciação de planos de usuários.

Outros trabalhos propõem a aproximação dos recursos computacionais da nuvem para perto do cliente, a fim de reduzir a latência e naturalmente melhorar a qualidade das aplicações sensíveis ao atraso e ao *jitter*. Em geral, esses trabalhos focam nos dispositivos móveis, redes de acesso sem fio, aplicações de tempo real e heterogeneidade.

Bonomi et al. (2012) propõe que a infraestrutura de nuvem permita mobilidade e interoperabilidade, e por tanto suporte técnicas como Locator/ID Separation Protocol (LISP) e federação. Trata-se de uma abordagem bem mais pervasiva, que inclui veículos conectados, *Smart Grid*, *Smart City* e *Wireless Sensors and Actuators Network* (WSAN).

Devido a uma necessária restrição de escopo neste trabalho, não são tratadas questões de mobilidade entre CSPs ou mesmo MECs, mas em teoria a possibilidade de *roaming* é aceitável no *framework* aqui proposto.

Satyanarayanan et al. (2009) justifica a necessidade da abordagem de nuvem próxima ao cliente com o argumento de que nos próximos anos a largura de banda tende a melhorar, enquanto a latência tende a se manter estática ou até mesmo piorar. É também uma abordagem ubíqua, na qual *cloudlets* podem ser colocadas em qualquer estabelecimento, como empresas, cafés e restaurantes e os dados presentes são exclusivamente dados efêmeros, como caches e códigos que estão disponíveis em outros locais. O objetivo é usar a computação móvel para aumentar as capacidades cognitivas do usuário, como reconhecimento facial e de fala, processamento de linguagem natural, aprendizagem de máquina, realidade aumentada, planejamento e tomada de decisão.

No presente trabalho é possível que *cloudlets* sejam criadas com a utilização de servidores de multimídia, uma implantação simplificada de um MEC, na qual representaria mais um dispositivo com o qual dividir o processamento. Entretanto, questões de segurança, como estabelecimento de confiança ou confiança baseado em reputação, ainda devem ser pensados (GARRISS et al., 2008; SURIE et al., 2007).

Os trabalhos citados anteriormente levantam estatísticas para demonstração de viabilidade por meio de experimentos limitados, feitos com simuladores de rede, ou ambientes controlados (rede local) ou mesmo usando provedores de nuvem e não máquinas na borda da nuvem como normalmente proposto.

Este trabalho também carece de experimentos mais elaborados, ao dar foco às facilidades oferecidas aos desenvolvedores de aplicações, demonstrando o funcionamento do *framework*. Pressupõe-se os bons resultados atingidos pelos trabalhos relacionados, que usam técnicas semelhantes, mas que não possuem a abrangência de recursos e funcionalidades desta proposta.

Alguns serviços já estão em funcionamento no que diz respeito a execução de jogos fora do ambiente do cliente entregues via *streaming* (*Game-as-a-Service* — GaaS), e que em teoria trata os mesmos tipos de problemas impostos pela interatividade. Dentre esse trabalhos podem ser citados NVIDIA GRID™ (NVIDIA, 2015) e Sony PlayStation™ Now (SONY, 2014). Outras empresas já tentaram fornecer tal serviço, como a OnLive e a Gaikai que foram adiquiridas pela Sony.

Tanto esses serviços como suas versões em rede local (*In-Home Streaming*), como o Steam In-Home Streaming (STEAM, 2014), NVIDIA GameStreaming™ (NVIDIA, 2014) e o Sony Remote Play (SONY, 2006), tem um funcinamento bem simples que se resumem em 3 etapas: (i) capturar os eventos de entrado; (ii) processá-los remotamente; e (iii) retornar um *streaming* de vídeo e áudio para o cliente.

Apesar de neste trabalho ser proposto um jogo como protótipo de aplicação usando o *middleware* que segue as características citadas, vale ressaltar que o *middleware* não está limitado a esse tipo de aplicação. Outro ponto relevante é que mesmos esses serviços podem tirar proveito da especificação da cadeia de valor apresentada aqui para aumentar a abrangencia de seus serviços com alta QoS. Por último esse trabalho propõe o *middleware* a fim de criar um ambiente pervarsivo usando os dispositivos do cliente independente de fabricante diferente do proposto, por exemplo, pelos NVIDIA® SHIELD™ (NVIDIA,

2013).

Por fim, este trabalho leva em consideração alguns trabalhos no que diz respeito às áreas de aplicações pervasivas (GRIMM et al., 2004), *middleware* adaptativo (SADJADI; MCKINLEY, 2003) e reflexão computacional (KON et al., 2002).

4 CADEIA DE VALOR

O *framework* proposto neste trabalho compreende (i) a definição de uma arquitetura capaz de oferecer os componentes necessários para uma infraestrutura de suporte à aplicações multimídia adaptativas em nuvem (Seção 4.1); (ii) a definição de um *workflow* que envolve as diversas etapas da provisão de um serviço (Seção 4.2); (iii) questões de QoS (Seção 4.3); e (iv) a especificação de um *middleware* que facilita o desenvolvimento de aplicações que explorem os recursos da nuvem e dos dispositivos em torno do cliente (Seção 6) como será visto no capítulo a seguinte.

4.1 VISÃO GERAL DA CADEIA DE VALOR

A arquitetura é dividida em 3 atores principais, não sendo necessária a presença de todos para um funcionamento mínimo[7]. Os atores são os seguintes: o *Cloud Service Provider* CSP responsável pelos *Media-Edge Clouds* (MEC), o *Multimedia Service Provider* (MSP) e o usuário. Ver figura 4.1.

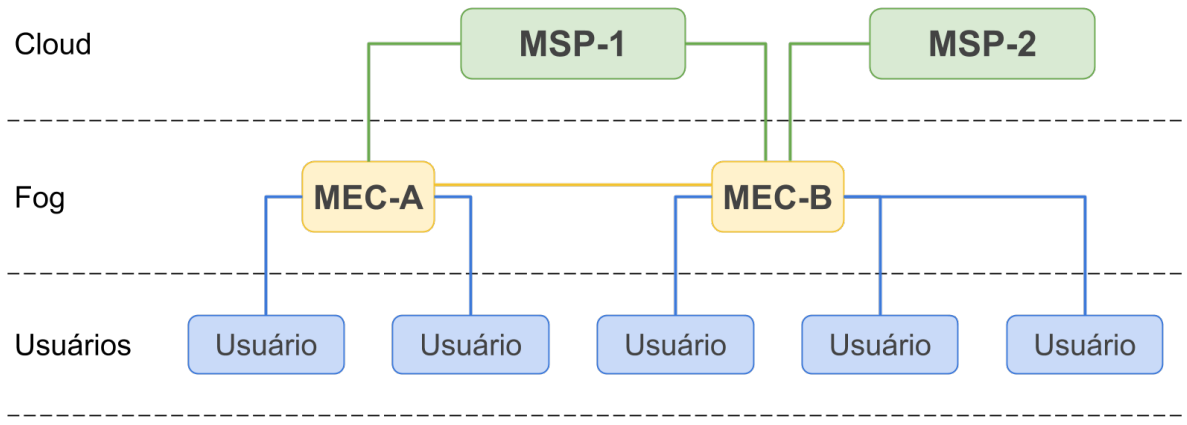

Figura 4.1: Cadeia de valor envolvendo MSP, CSP (MECs) e usuários

O MEC envolve todos os módulos necessários para lidar com o *cluster*, garantia de QoS, adaptação de conteúdo, bem como alguns algoritmos padrões, por exemplo, algoritmos de negociação. Fornece também uma *Application Programming Interface* (API) bem definida permitindo a criação de um *front-end* para gerenciamento personalizado. Esse

[7] O cliente por si só é capaz de executar uma aplicação independente de nuvem e completamente funcional

componente pode ser usado por um *Network Service Provider* (NSP) na borda da nuvem, ou mesmo como um provedor à parte, em cima de um NSP, ou seja, o CSP.

O MSP é intencionado para provedores de conteúdo que desejam entregar serviços de multimídia com alta QoE para seus clientes. O *front-end* do MSP inclui funcionalidades pré-definidas, como adicionar/remover MECs e políticas que serão usadas pelo cliente, mas pode ser personalizado com base em sua API. Ver Figuras 4.2 e 4.3.

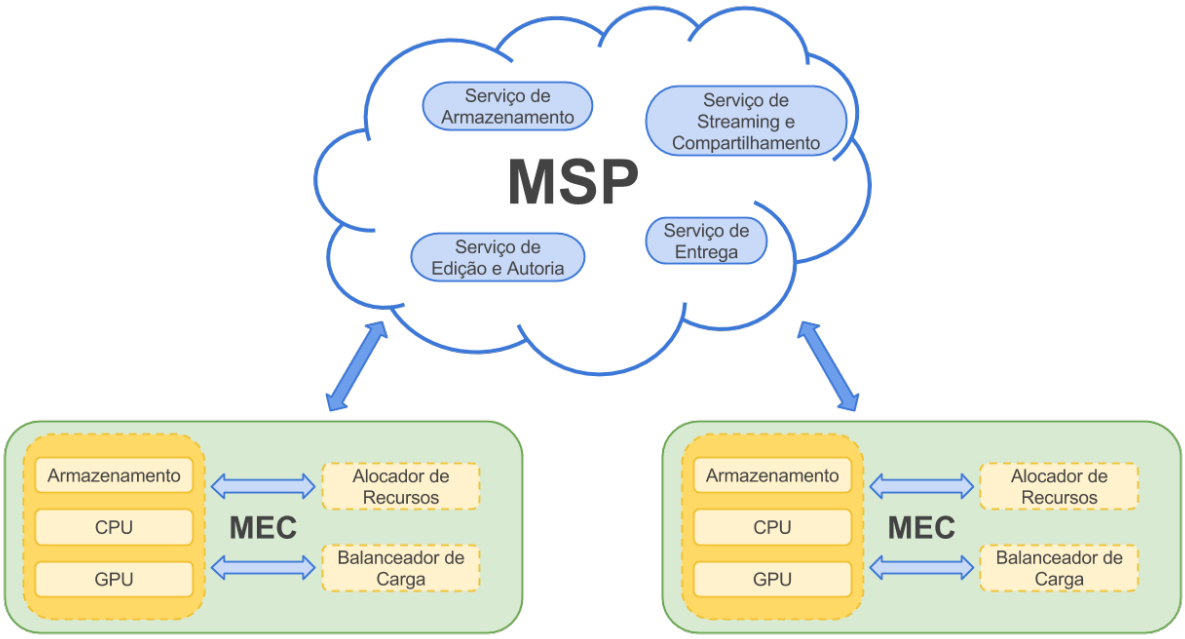

Figura 4.2: MSP usando múltiplos MECs

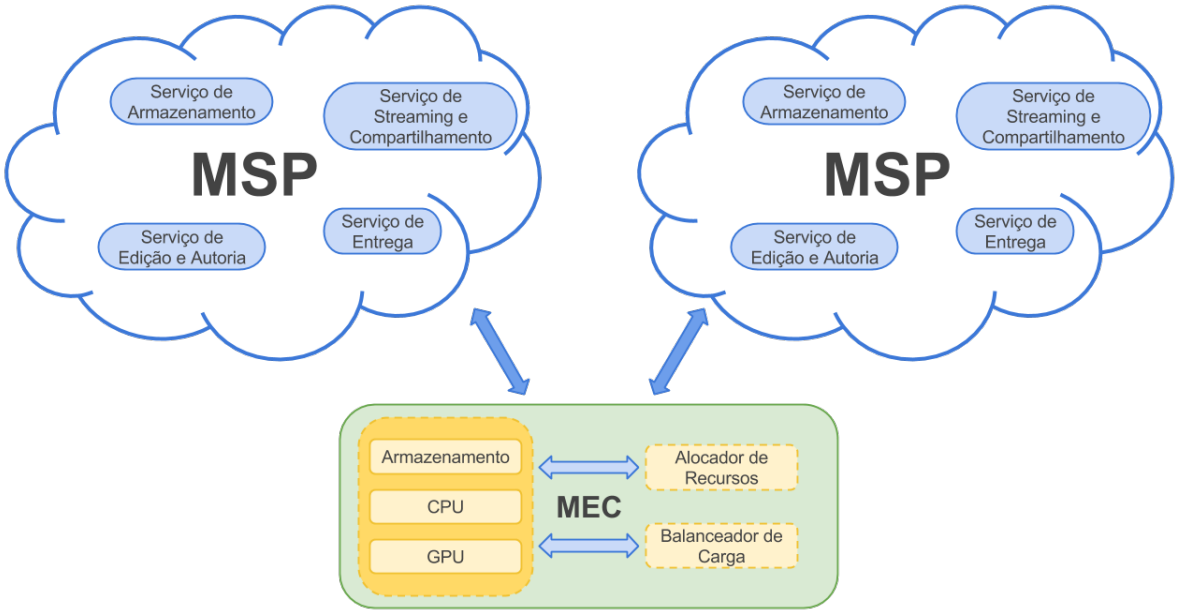

Figura 4.3: MSPs compartilhando MEC

O lado do cliente contempla funcionalidades de migração de tarefas e dados, negociação, descoberta, etc. Essas funcionalidades foram pensadas para serem usadas na criação das aplicações pelos desenvolvedores e não necessariamente precisam de qualquer interação com a nuvem, podendo executar exclusivamente na rede do usuário envolvendo seus variados dispositivos.

A aplicação no cliente é modularizada, onde cada módulo é uma tarefa que pode ser delegada a um dispositivo diferente e o resultado dessa tarefa é armazenado em um objeto que por sua vez é encaminhado para a próxima tarefa. Para fins de referência esse objeto é denominado Objeto Nômade e cada tarefa cria um desse sempre que deseja enviar informação para a próxima.

A aplicação pode contemplar uma interface feita para estender as funcionalidades a um ambiente externo (Internet) tratando questões como endereços de escopo local, *firewalls*, etc.

4.2 WORKFLOW DE PROVISÃO DO SERVIÇO

Será mostrado a seguir como os componentes da arquitetura envolvendo MSP, CSP e cliente interagem, estabelecendo um *workflow* que se estende nas diversas fases de provisão do serviço multimídia, compreendendo a solicitação, negociação, estabelecimento, manutenção e encerramento do serviço (MORENO, 2002).

4.2.1 SOLICITAÇÃO DO SERVIÇO

Durante a solicitação de um serviço, o MSP deve decidir qual MEC será usado para auxiliar o processamento do usuário. Uma vez que o usuário inicia a aplicação, essa se conecta ao MSP onde opcionalmente será requisitada a autenticação. O primeiro passo é identificar qual política será usada para fornecer recursos computacionais para o cliente. Essa política pode variar de acordo com o plano do usuário, o tipo de aplicação, o contrato entre o MSP e CSP mais próximo, dentre outros fatores.

O MSP deve descobrir dentre os MECs de um ou mais CSPs qual vai atender o usuário. É possível que o usuário não esteja na rede de nenhum desses MECs, cabendo ao MSP decidir se continuará a fornecer a aplicação mesmo sem garantia de QoS. Outro possível problema de QoS pode surgir de aplicações multiusuário onde esses se encontram

em MECs diferentes ou até mesmo CSPs diferentes. Nesse caso o MSP pode verificar se existe algum mecanismo ou acordo para garantir a QoS entre as redes.

Identificado o MEC e a política, a aplicação cliente deve ser informada das decisões. A política será levada em consideração na fase de negociação com o MEC. A partir daqui o cliente deve se conectar ao MEC designado e compartilhar com ele a política do MSP. Devem ser aplicadas técnicas para garantir a autenticidade da política, como assinatura digital. Ver Figura 4.4.

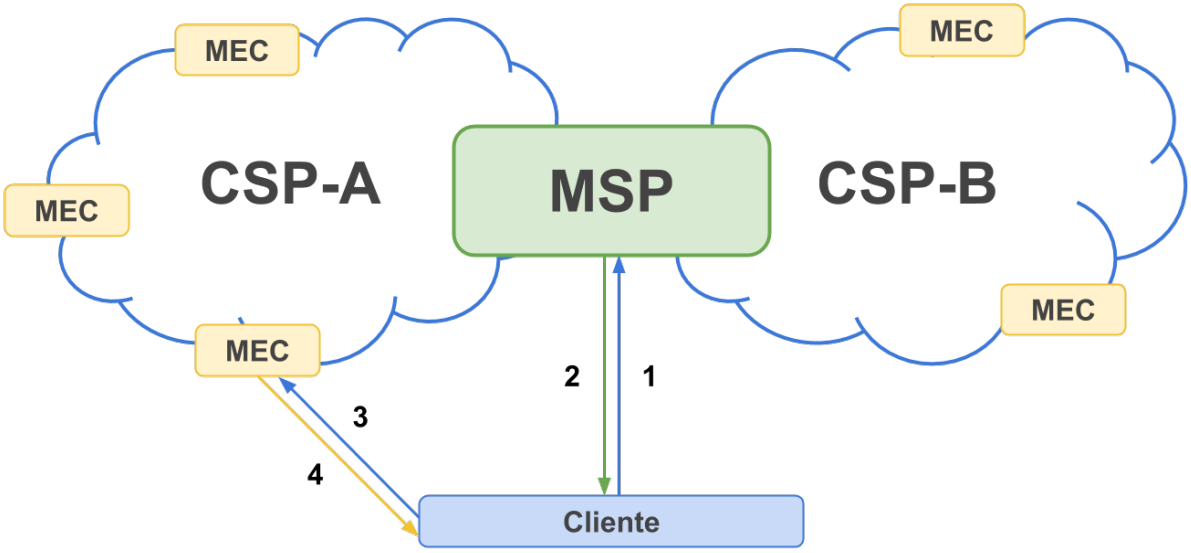

Figura 4.4: Fluxo de conexão: (1) cliente se conecta ao MSP e requisita uma aplicação; (2) MSP identifica e envia a política adequada para o usuário; (3) inicia a negociação com o MEC usando a política do MSP; (4) responde a negociação e estabelece o serviço.

4.2.2 NEGOCIAÇÃO DE RECURSOS

A negociação de recursos será explicada em mais detalhes e melhor exemplificada na Seção 5.3.3. Em resumo, é durante a negociação que são acionados mecanismos para a tomada de decisões de alocação de recursos e adaptação. Tais mecanismos também podem ser usados na sintonização do serviço (Seção 4.2.4).

A negociação pode levar em consideração recursos de *hardware* e *software*, tanto os descritos de forma quantitativa quanto os de forma qualitativa. Recursos quantitativos são aqueles que podem ser quantificados, por exemplo, tamanho de memória e capacidade de armazenamento. Recursos qualitativos são aqueles cuja especificação é dado como existente ou não existente em um dispositivo e podem ser tanto recursos de *software* quanto de *hardware*, por exemplo, se o dispositivo possui ou é um periférico de entra/saída

e se determinado codec (*coder-decoder*) está instalado. A negociação também pode levar em consideração o contexto desses recursos, o funcionamento da aplicação e regras de negócio de cada um dos atores envolvidos, por exemplo, desenvolvedor, MSP e CSP.

4.2.3 ESTABELECIMENTO DO SERVIÇO

Assim que a negociação termina de forma bem-sucedida, decidindo quais recursos serão alocados no caminho entre cliente e MEC, vem a etapa de alocação propriamente dita. Em suma o MEC pode simplesmente trabalhar com *best effort* ou prover mecanismos de reserva de recurso. Existem técnicas para aumentar a QoS mesmo sobre um serviço de *best effort*. Quando usando reserva de recursos, não só o MEC sob responsabilidade do CSP o deve fazer para garantir a QoS da aplicação, mas também deve ser requisitada a reserva de recursos no NSP. Afinal, o retardo e o jitter na rede estão entre os maiores limitadores de aplicações interativas (Seção 8.1).

4.2.4 MANUTENÇÃO DO SERVIÇO

Após o estabelecimento do serviço, a aplicação e o ambiente devem ser constantemente monitorados para reagir de forma rápida e eficaz às condições indesejadas ou possíveis otimizações, usando o mecanismo de sintonização, similar à negociação, promovendo re-negociações de recursos. Condições indesejadas podem incluir anomalias na rede, esgotamento de recursos ou problemas no dispositivo cliente, infrações nas políticas do MSP ou do MEC.

Exemplificando, em uma situação em que a latência da rede aumente ao ponto do tempo gasto para enviar e receber dados tornar-se maior do que o tempo para processar localmente, nesse caso todas as tarefas devem migrar para o dispositivo local. Outro cenário é o caso de aplicações e serviços em *background* começarem a consumir recursos que antes estavam dedicados a aplicação em foco, de tal forma que o dispositivo não seja capaz de atingir a experiência esperada. Nesse caso, a migração de algumas tarefas para a nuvem pode resolver o problema. Como último exemplo, é possível a situação em que os módulos da aplicação executando em nuvem comecem a consumir mais recursos do que o limite reservado para o cliente, o que implica infringir a política, e como solução a responsabilidade do processamento de alguma tarefa pode ser transferida para o cliente.

Esses foram exemplos de condições indesejadas, entretanto a sintonização pode acontecer para atingir condições mais favoráveis, por exemplo, se um usuário está interagindo em um dispositivo móvel a caminho de casa, com uma determinada resolução limitada pela política da nuvem, ao chegar em sua casa, seu computador de mesa pode ser descoberto automaticamente, permitindo a migração de tarefas para esse e assim aumentando a qualidade da aplicação.

Tanto na manutenção do serviço, quanto na etapa de negociação de recursos, os desafios se resumem em identificar as necessidades da aplicação, os recursos disponíveis, calcular o melhor cenário para a aplicação e então executar a migração respeitando as restrições determinadas pelas políticas.

4.2.5 ENCERRAMENTO DO SERVIÇO

O encerramento de um serviço consiste em liberar os recursos da nuvem. Isso pode ser feito de maneira explicita informando tanto MEC quanto MSP, ou pode ser feito de maneira automática identificando um período de inatividade.

4.3 QUALIDADE DE SERVIÇO

4.3.1 DEFINIÇÃO DE QUALIDADE COMO UM SERVIÇO (QAAS)

Para se conseguir uma real garantia de QoS, todo o ambiente deve ter essa capacidade, ou seja, não só o núcleo da rede, mas também os sistemas finais devem garantir a disponibilidade de processamento, memória, rede e etc (MORENO, 2002). Porém, atualmente a maior parte dos sistemas domésticos são de propósito geral inviabilizando isso. Como alternativa, a arquitetura tenta fornecer qualidade através da renegociação dinâmica na presença de anomalias e adicionalmente oferece uma interface para que o CSP possa requisitar ao NSP garantia através de algum mecanismo usado pelo mesmo.

Esses mecanismos podem variar de acordo com a rede e interesse do provedor, por exemplo, dependendo da rede a comutação de circuitos virtuais pode ser uma alternativa com mais qualidade do que a rede de comutação de pacotes. Mesmo em redes de comutação de pacotes é possível oferecer maior qualidade de serviço do que o simples *best effort*. Para tal é possível usar do modelo de *Integrated Services* (IntServ) com protocolos como o *Resource Reservation Protocol* (RSVP) ao invés de *Differentiated Services* (DiffServ).

Técnicas de reserva de recurso são dispendiosas, pois no caso de ociosidade do *link* ele continua dedicado se tornando um recurso desperdiçado. Nesse ponto a *Software-Defined Networking* (SDN) pode fornecer algo muito mais adaptável com uso de QoS dinâmica baseado na aplicação.

Vale lembrar que se trata da conexão entre o cliente e o MEC na borda da rede, uma infraestrutura que normalmente está sob o domínio de uma única empresa, tornando possível a implementação de qualquer técnica sem muitos problemas burocráticos. A garantia de qualidade entre MECs será discutida na Seção 4.3.2.

Além da reserva de *link* o MEC foi idealizado para naturalmente aumentar a QoS com serviços multimídia, isso devido a separação em *clusters* que se ajustam aos tipos específicos de aplicações. Por exemplo, uma aplicação que requer mais computação gráfica pode ter recursos reservados no *cluster* de GPU enquanto uma aplicação tradicional pode ter recursos reservados no *cluster* de CPU.

Independente da alternativa usada para prover QoS, o custo para o Service Provider (SP) aumentará. Para tornar de interesse desses, é sugerido que o custo seja repassado para o cliente com um modelo de cobrança da QoS como um serviço, seguindo a ideia do *utility computing* a fim de viabilizar a implantação de tal serviço, o qual denominamos *Quality-as-a-Service* (QaaS). Esse serviço é uma abstração do gerenciamento como um serviço, ou seja, MaaS. Ver Figura 4.5.

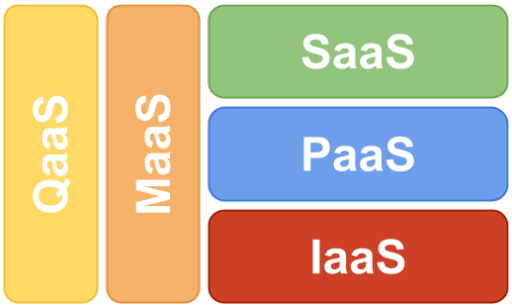

Figura 4.5: Quality-as-a-Service (QaaS)

4.3.2 SUPORTE ALÉM DA BORDA

Aplicações onde apenas um usuário interage funcionam bem se limitando a borda da nuvem, ou seja, cliente e MEC. Esse tipo de interação é a mais simples, pois limita-se normalmente ao domínio administrativo de uma única organização facilitando aplicação de técnicas de QoS. E pela simples limitação geográfica o tempo de resposta tende a estar dentro do aceitável, proporcionando a possibilidade de aplicações interativas serem processadas fora da rede do usuário.

Entretanto, aplicações com múltiplos usuários devem ser pensadas com mais cuidado.

Imagine o cenário onde três usuários distantes geograficamente e sob a gerência de provedores distintos querem participar da mesma partida de um jogo e um deles será apenas um expectador. Considere também que esse jogo é dividido em 3 etapas, onde a primeira é a captura de teclas do jogador, a segunda é o processamento do resultado na forma de *streamings* de vídeo e áudio e a terceira a exibição desses *streamings*. Como última informação, a etapa de processamento é executada na nuvem (MEC). Ver Figura 4.6.

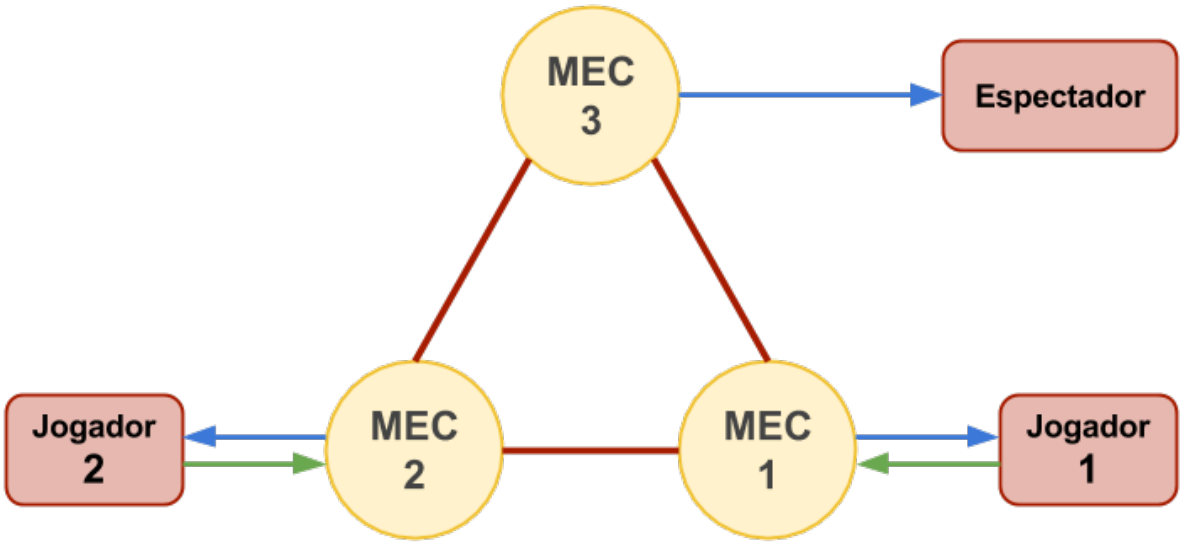

Figura 4.6: Aplicação executando entre MECs

Para cada usuário a degradação do serviço ocorrerá principalmente se houver atraso entre pressionar uma tecla e visualizar o resultado. Esse processo não será afetado, uma vez que continua basicamente o mesmo que em uma partida *single player*, ou seja, a etapa de captura dos eventos de entrada é realizada, o resultado é enviado para a borda da nuvem processar o estado do jogo gerando os *streamings* de vídeo e áudio que então voltam para o jogador exibi-lo. Porém, agora o estado do jogo deve ser sincronizado com o outro jogador, bem como os *streamings* dessa partida devem ser enviados para o expectador.

A sensação de atraso ainda será sentida se a comunicação entre os jogadores tiver a latência muito alta. Já com o expectador o atraso dos *streamings* é irrelevante, tendo como principais problemas o *jitter* e o *skew*[8]. Pode-se notar que há requisitos diferentes de qualidade que devem ser garantidos na comunicação entre provedores para a boa experiência da aplicação. Portanto, uma aplicação com múltiplos usuários de provedores

[8]Skew é a diferença entre os tempos de chegada de diferentes mídias que deveriam estar sincronizadas.

diferentes só pode ser provida (com qualidade) se há um acordo entre esses provedores.

Mesmo entre MECs do mesmo CSP pode ser difícil de prover QoS, pois esse pode não ser também o NSP e precisará aumentar seus gastos contratando mais *link* para prover comunicação de qualidade. Para motivar o CSP é ressaltada a necessidade de fornecer QoS como um serviço repassando os gastos para o cliente e motivando a constante melhoria do serviço.

O MSP é o responsável por avaliar a disponibilidade de uma aplicação para o cliente de acordo com os MECs envolvidos.

5 ARQUITETURA

É tomado como base a arquitetura proposta em Zhu et al. (2011), onde é definido o MEC com o propósito de atingir alta QoS e o MSP com o propósito de entregar serviços multimídia de alta QoE fazendo uso do *hardware* especializado no MEC. O MSP foi proposto tendo em vista a perspectiva de multimídia ciente da nuvem enquanto o MEC é uma perspectiva da nuvem ciente de multimídia. Esses conceitos são mesclados na arquitetura proposta nesse trabalho a qual nos referimos como nuvem multimídia.

Nas seções a seguir serão detalhados os componentes MSP, MEC e por fim o *middleware* que é a base para a aplicação rodando no cliente e no MEC.

5.1 MSP — MULTIMEDIA SERVICE PROVIDER

Para uma melhor compreensão do fluxo de negociação é necessário uma visão mais detalhada do núcleo do MSP. Ver Figura 5.1.

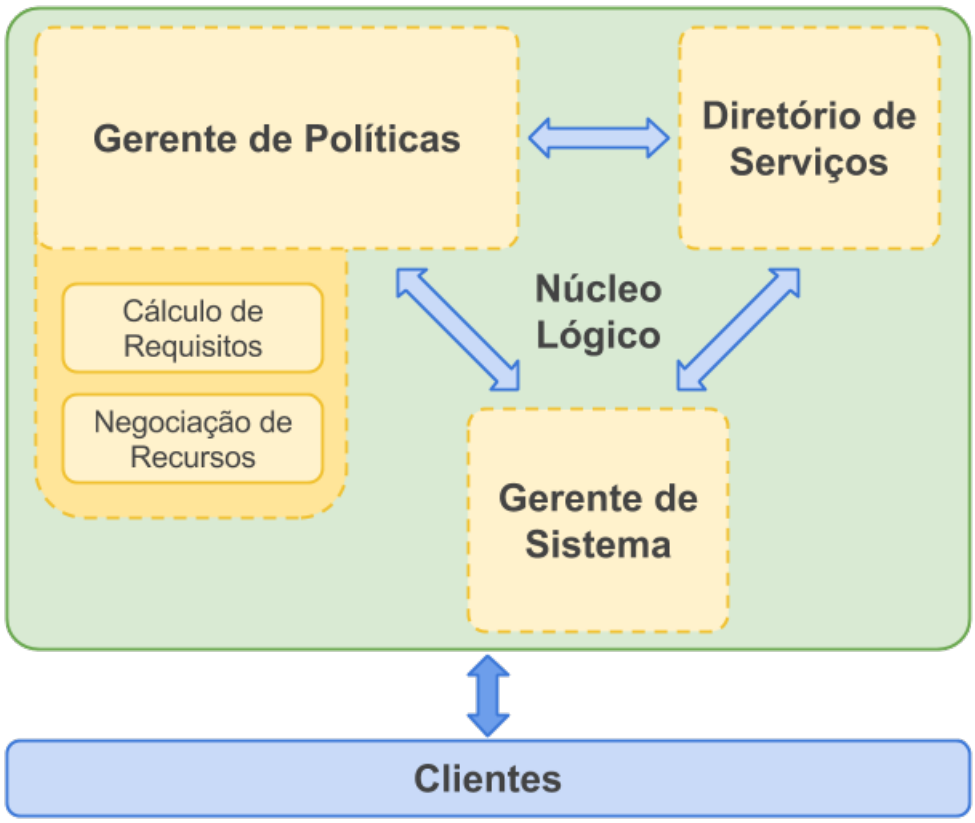

Figura 5.1: Núcleo do Multimedia Service Provider (MSP)

Quando a aplicação deseja iniciar a comunicação com a nuvem ela deve enviar uma requisição com informações como, identificação, aplicação desejada, descrição do dispositivo entre outras. É possível que o MSP forneça uma interface para que a escolha da aplicação seja feita e então a requisição com os itens informados acima é enviada.

5.1.1 GERENTE DE SISTEMA

Ao chegar no MSP, a requisição passa pelo módulo Gerente de Sistema que se encarrega de algumas burocracias como o processo de autenticação e autorização. A próxima tarefa é identificar se o usuário está em uma área de cobertura de algum MEC associado ao MSP.

Uma vez identificado o usuário e o MEC, o Gerente de Sistema verifica qual a aplicação desejada e repassa essas informações para o módulo Gerente de Políticas.

5.1.2 GERENTE DE POLÍTICAS

O Gerente de Políticas é dividido em 2 etapas, são elas, descobrir os requisitos necessários para uma aplicação e negociar com o cliente baseado nas regras de negócio.

O submódulo de Cálculo de Requisitos usa o módulo Diretório de Serviços para identificar a aplicação e seus requisitos. Com essa informação o fluxo continua no submódulo de Negociação de Recursos que junto à identificação do usuário e MEC verifica qual algoritmo usar. O algoritmo pode ser escolhido levando em conta vários fatores técnicos e de negócio, por exemplo, o tipo de aplicação e plano contratado pelo usuário. O resultado dessas etapas é enviado ao usuário que depois repassa para o MEC selecionado.

5.1.3 DIRETÓRIO DE SERVIÇOS

É um repositório com as aplicações fornecidas pelo MSP e seus respectivos requisitos de *hardware* e *software*. Esses requisitos podem ser especificados manualmente ou automaticamente e devem estar em harmônia com a descrição de recursos dos dispositivos clientes.

5.2 MEC — MEDIA-EDGE CLOUD

O MEC funciona como uma *Content Delivery Network* (CDN) na borda da nuvem, porém não se limita à entrega de conteúdo, ou seja, o armazenamento de dados. Esse é composto por *clusters* de armazenamento, CPUs e GPUs. Outro item descrito por Zhu et al. (2011) é o *Load Balancer*, também referido como *Program Tracker* para os *clusters* de CPU e GPU e *Data Tracker* para o *cluster* de armazenamento. Ver Figura 5.2.

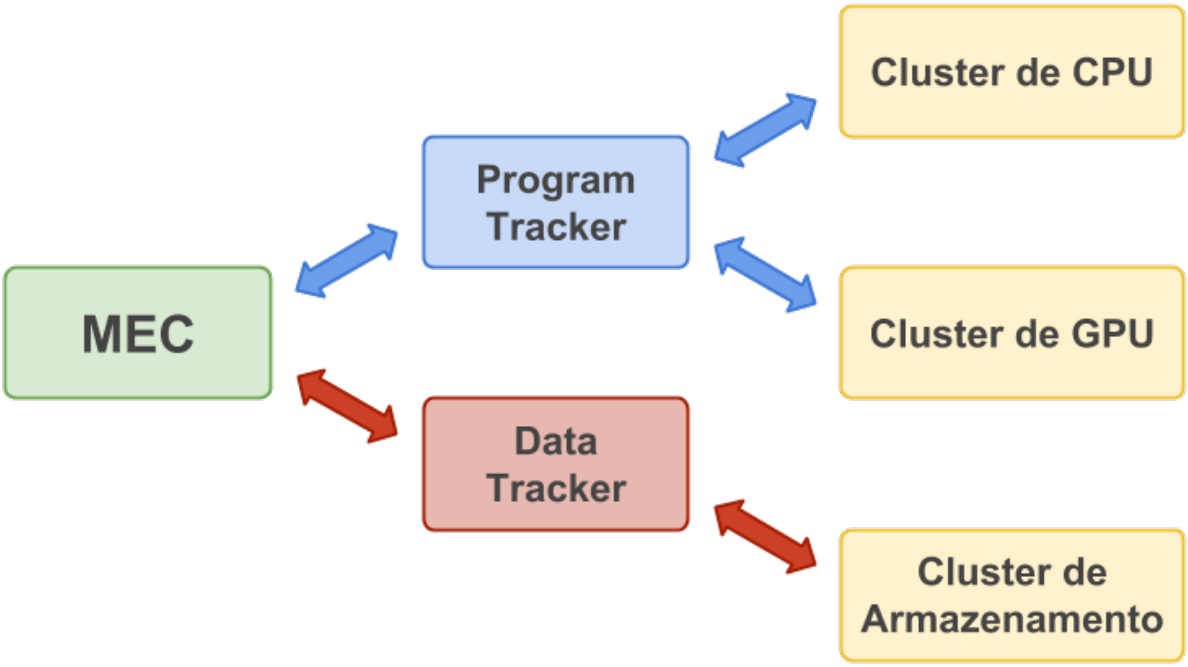

Figura 5.2: Separação do Load Balancer em Program Tracker (clusters de CPU e GPU) e Data Tracker (cluster de armazenamento)(ZHU et al., 2011)

Para o propósito desse trabalho mais um componente foi adicionado, o núcleo lógico, responsável por gerenciar toda a comunicação com as aplicações no cliente e em outros MECs. Ver Figura 5.3.

5.2.1 CLUSTERS

É esperado que cada CSP possa escolher como montar o seu MEC, seja na escolha do *hardware* ou do *software*. Inicialmente o *hardware* para o *cluster* foi pensado em cima do conceito de *Network Functions Virtualization* (NFV), mas nada impede que escolhas diferentes sejam feitas por provedores diferentes. Também para o *software*, qualquer biblioteca pode ser usada para o gerenciamento do *cluster*.

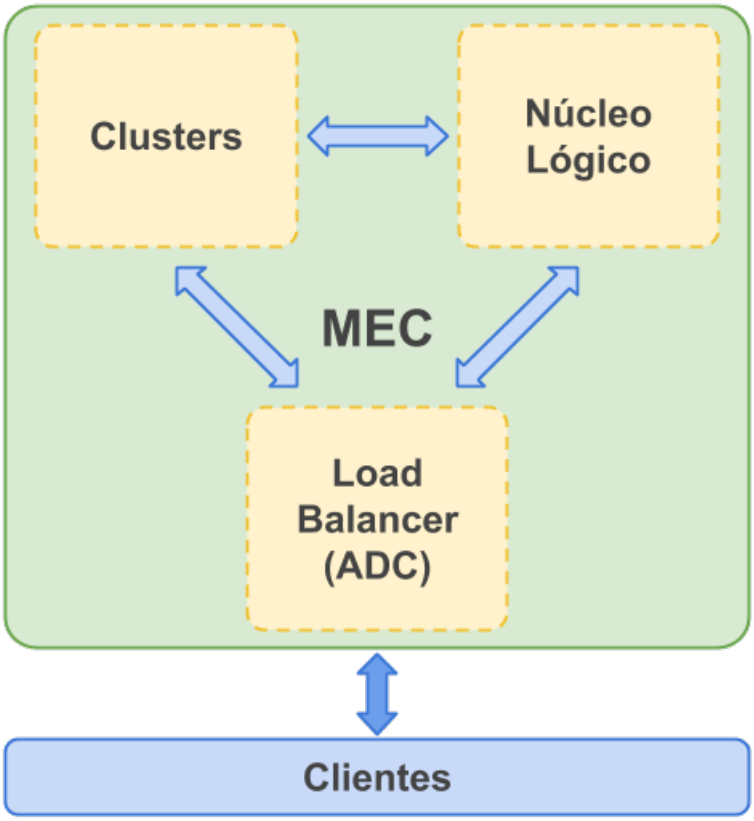

Figura 5.3: Núcleo do Media-Edge Cloud (MEC)

O MEC por definição deve ficar na borda da nuvem, mas em teoria é possível a adoção de uma nuvem pública em vez da construção de um *cluster*. Nesse caso alguns fatores devem ser levados em conta, como o tempo de resposta para aplicações interativas. Dependendo da localização da nuvem pública, da garantia de qualidade do tráfego entre os envolvidos e da natureza da aplicação isso pode ser perfeitamente possível.

5.2.2 LOAD BALANCER

Nesse trabalho é proposto a substituição do *Load Balancer* por um *Application Delivery Controller* (ADC) (SALCHOW, 2014). Os ADCs se baseiam nos conceitos do *Load Balancer* e adicionam alguma inteligência. Algumas características oferecidas por eles são:

- Compressão
- Cache
- Multiplexação de conexões
- *Traffic shaping*

- *Firewall* de aplicação
- *Secure Sockets Layer* (SSL) *offloading*
- *Switch* de conteúdo
- Proteção contra *Distributed Denial of Service* (DDoS)
- Avançadas estratégias de roteamento
- Monitoramento de saúde de servidor
- *Dynamic Site Acceleration* (DSA)
- *Front-End Optimization* (FEO)
- Aceleração de conteúdo de mobile
- Aceleração de *streaming* de vídeo

Essas funcionalidades podem proporcionar melhorias consideráveis na entrega de conteúdos multimídia, tanto que o ADC é uma tecnologia já muito usada nas CDNs. É importante salientar que assim como a escolha do *cluster* é independente da arquitetura, o *Load Balancer* também é, apesar da sugestão em se usar o ADC que permite na prática a integração do núcleo lógico nesse componente.

5.2.3 NÚCLEO LÓGICO

Como foi visto, o propósito do MEC é dividir tarefas com os clientes e prover alta QoS. Esse é o módulo responsável por executar a negociação com o cliente e suas funcionalidades são similares as encontradas do lado cliente.

5.3 CLIENTE

Os principais módulos do lado cliente estão em torno dos desafios para a divisão de tarefas que se resumem em: (i) identificar os recursos disponíveis em um dispositivo (Descoberta de Recursos); (ii) identificar os requisitos de uma aplicação (Descoberta de Requisitos); (iii) decidir a divisão entre nuvem e cliente (Negociação de Recursos); (iv) dividir a aplicação em módulos (Núcleo).

5.3.1 DESCOBERTA DE RECURSOS

Esse é o módulo responsável por coletar as informações do dispositivo, sendo que cada dispositivo executa um e apenas um processo desse módulo independente de quantas instâncias[9] da mesma aplicação ou de aplicações distintas estejam executando nele.

O nível de detalhamento deve variar de acordo com a necessidade da aplicação, podendo ser escalável. É importante fornecer uma maneira clara de notificar o usuário sobre quais informações estão sendo coletadas.

As informações coletadas são os dados de entrada para realizar a negociação de tarefas. Dentre as informações pode-se ter as características do dispositivo e sua rede, bem como o contexto de ambas, ou seja, se a rede está congestionada, se a latência está alta ou se a memória do dispositivo está baixa.

O contexto muitas vezes pode ser mais importante para conseguir aumentar a QoS do que a especificação. Essa especificação pode conter recursos quantitativos e qualitativos. Essas informações, especificação e contexto, permitem que a negociação seja realizada. A seguir é mostrado uma lista de possíveis informações contidas na descrição de capacidades do dispositivo.

- **Hardware:** CPU, memória, GPU, armazenamento, resolução de tela;
- **Contexto de hardware:** uso de CPU, memória, GPU, bateria;
- **Qualidades do hardware:** periféricos de entrada/saída;
- **Qualidades do software:** formatos suportados de áudio, vídeo e imagem;
- **Especificação de rede:** largura de banda, meio físico;
- **Contexto de rede:** uso da largura de banda, latência, *jitter*.

5.3.2 DESCOBERTA DE REQUISITOS

A descrição dos requisitos da aplicação é a contraparte da descrição dos recursos do dispositivo e como tal segue a mesma estrutura. Essas descrições devem ser padronizada para que diferentes módulos da arquitetura possam se intercomunicar, ou seja, o código para captura dos recursos do dispositivo e os requisitos da aplicação especificado pelo

[9]Uma instância é uma aplicação em execução em um determinado dispositivo, em geral um processo.

desenvolvedor ou descobertos automaticamente devem ser inteligíveis pelo algoritmo de negociação.

5.3.3 NEGOCIAÇÃO DE RECURSOS

Nos dias de hoje, os serviços fornecidos pela computação em nuvem, em sua grande maioria executam exclusivamente em nuvem, o que implica alguns problemas e desperdícios. A arquitetura proposta visa resolver ou amenizar esses problemas por incluir ambos MSP e cliente na cadeia de processamento. Para essa tarefa o principal mecanismo é o de negociação e suas dependências.

O *framework* foi concebido para ser distribuído, não precisando de um ponto central de gerência. Sendo assim, a aplicação que explora o *framework* pode fazer uso da nuvem ou apenas executar localmente. O mecanismo de negociação pode ser trabalhado em níveis de abstração diferentes para atender cada cenário.

Por exemplo, torna-se possível o cenário onde o usuário adquiriu uma aplicação e deseja executá-la localmente, porém distribuída entre seus dispositivos. Essa aplicação pode usar o mecanismo de negociação para simplesmente verificar se o dispositivo em questão é capaz de executar uma tarefa, ou seja, se possui os requisitos de *hardware* e/ou *software* necessários. Uma vez que os dispositivos estão inteiramente sob o controle do usuário, não há necessidade de envolver regras de negócio, sendo as limitações técnicas o único empecilho.

Para tornar mais claras quais são essas limitações técnicas, toma-se como ilustração uma aplicação dividida em 3 módulos de execução: (i) captura de eventos de entrada; (ii) processamento do estado da aplicação; (iii) exibição do resultado.

O requisito de *hardware* para a primeira tarefa é que exista um periférico de entrada, podendo variar de um tradicional teclado ou controle de jogo, até algum sensor, como um acelerômetro ou sensor de fumaça; o requisito da segunda tarefa é que tenha capacidade de processamento, podendo ser um *cluster* de CPUs ou GPUs de acordo com a aplicação; por fim, o requisito para a terceira tarefa pode ser a necessidade de uma tela para exibição ou mesmo um local para armazenar o resultado do processamento.

Para uma aplicação simples, a necessidade de um *cluster* obviamente não é o caso, então a especificação de requisitos do módulo de processamento podem ser mais minuciosos, por exemplo, definindo a quantidade mínima de núcleos de CPU, de *Millions of*

Instructions Per Second (MIPS) ou *Floating-point Operations Per Second* (FLOPS). Esse nível de granularidade muitas vezes não é possível em sistemas operacionais de propósito geral e nem mesmo interessante devido à complexidade de se realizar a especificação, seja manualmente ou automaticamente.

Por último, sendo a etapa final implementada pelo módulo de exibição de vídeo, essa pode ter como requisito o suporte a algum tipo de codificação específico que pode ser negociado entre essa etapa e sua predecessora baseado nos recursos disponíveis. Esse conceito é conhecido como adaptação de conteúdo.

Com esse exemplo se pode perceber que os recursos podem ser quantitativos ou qualitativos, onde a primeira faz referência a quanto de um determinado recurso cada computador possui e a segunda determina se um computador possui ou não o recurso disponível, seja *hardware* ou *software*.

Em um cenário onde a nuvem é incluída pode-se ampliar o mecanismo de negociação para tomar decisões não se baseando apenas em questões técnicas, mas também em regras de negócio (ver Seção 5.3.6). Quando um usuário contrata os serviços da nuvem, essa se torna mais um nó na cadeia de processamento e como tal possui a autonomia para decidir se vai ou não processar determinada tarefa baseado em suas políticas. Ao envolver a nuvem é necessário o uso do módulo Gerente de Políticas para lidar com as regras de negócio da nuvem.

Vamos imaginar dois cenários diferentes para explorar as capacidades da arquitetura. No primeiro cenário o usuário adquire uma aplicação seguindo as 3 etapas citadas anteriormente, que a priori é processado *offline*, ou seja, sem qualquer participação da nuvem.

Em determinado momento, a nuvem é incluída para processar o estado da aplicação, mantendo como sua própria responsabilidade a captura dos eventos de entrada e a exibição do resultado. A motivação para isso podem ser diversas, por exemplo, lentidão devido a pouca capacidade de processamento do *hardware* ou porque o usuário pretende mudar para um dispositivo móvel e quer reduzir o consumo de bateria. Nesse caso a participação da nuvem é a de simplesmente fornecer processamento para a aplicação de acordo com as regras de negócio do serviço contratado.

No segundo cenário o usuário contrata a mesma aplicação como um serviço (SaaS). A diferença agora é que o usuário não adquiriu a aplicação, e sim está consumindo-a sob demanda.

Basicamente o que muda são as regras de negócio, por exemplo, supondo que o usuário queira processar tudo, o MSP pode vetar, pois o negócio desse se baseia na venda de propagandas que são embutidas na segunda etapa do processamento (STRIDE, 2015). Sendo assim o usuário pode participar apenas da primeira e última etapa. Nesse caso a aplicação instalada no dispositivo do usuário pode se limitar ao necessário para a execução dessas etapas.

Observe que uma aplicação simplificada também pode ser disponível para dispositivos com funcionalidades limitadas, ou seja, se o dispositivo é um *gamepad*, a aplicação pode se limitar ao código necessário para a captura dos eventos de entrada, já se o dispositivo é um monitor ou televisão, a aplicação pode se limitar ao código necessário para exibir o resultado.

Toda essa capacidade de negociação e distribuição é particularmente interessante se pensarmos na heterogeneidade, por exemplo, dispositivos móveis comparados com *clusters* em nuvem possuem capacidades de processamento e armazenamento irrisórias, mas os dispositivos móveis costumam ter sensores que não estão presentes na nuvem, permitindo criar maneiras inovadoras de se interagir com a aplicação e aumentar a imersão na mesma.

Outro ponto de desperdício comum em aplicações inteiramente em nuvem é o *hardware* do cliente. Como exemplo temos o cenário onde a aplicação em nuvem que o cliente está interagindo é um jogo e esse cliente possui uma placa de vídeo de última geração. Mesmo com essa combinação aparentemente perfeita o jogo está com uma resolução baixa, pois devido a alguma limitação de *hardware* ou política, a nuvem não pode processar nada acima disso e também não faz uso da placa de vídeo do cliente. Com o *framework* proposto a solução é simplesmente delegar a etapa de renderização para o cliente.

Para embasar essa questão de desperdícios pode-se olhar para os sistemas em grade que são projetados para usarem justamente os recursos subutilizados dos dispositivos normalmente residenciais e conectados a Internet.

Outro ponto importante na negociação diz respeito a resiliência. Por exemplo, caso a rede fique instável ou o MSP fique sobrecarregado haverá degradação do serviço executando exclusivamente em nuvem. Para sanar esse problema é proposto a reconfiguração dinâmica na presença de anomalias na nuvem e/ou rede. Problema na rede não necessariamente significa a completa desconexão, podendo ser apenas limitação de banda ou retardo. A renegociação de tarefas ou adaptação do conteúdo pode ser eficiente em aumentar a

QoE.

Esse problema já é tratado por algumas aplicações, por exemplo, editores de texto que ao perderem a conexão com o servidor continuam funcionando localmente e quando retomam a conexão sincronizam seu estado com a nuvem.

Dessa observação sobre mal funcionamento na rede pode-se perceber outros dois possíveis fatores de negociação, o contexto dos recursos e o funcionamento da aplicação. O contexto diz respeito a quantidade disponível de um recurso, indo além da quantidade absoluta. O funcionamento da aplicação pode ser visto como uma avaliação de nível mais alto de abstração, por exemplo, se a aplicação for um vídeo ou um jogo desejamos uma taxa mínima de *Frames Per Second* (FPS).

A coleta das informações pertinentes à negociação pode ser feita por meio de agentes de *software* em 3 níveis. Um agente atua em nível de sistema (Seção 5.3.6), que contém as regras de negócio e atua sobre todos os dispositivos na rede local envolvidos na aplicação em questão. Um segundo agente atua em nível de máquina (Seção 5.3.1), sendo responsável pela coleta de informações de *hardware* e *software*. O último agente atua em nível de instância da aplicação (Seção 5.3.7) para verificar o funcionamento individual de cada uma. Esse último é importante, pois o *framework* permite que uma mesma máquina execute mais de uma instância.

5.3.4 MIGRAÇÃO DE TAREFAS

A negociação por si só não é capaz de migrar uma tarefa, ela apenas analisa as capacidades dos dispositivos associados para tomar uma ação apropriada alinhada com os interesses dos *stakeholders*, por exemplo, administradores da nuvem, desenvolvedores e usuários da aplicação. A migração em si, é uma dessas ações que podem ser tomadas, um exemplo de outra ação simples é a negação do serviço.

A migração em geral é a movimentação ou deslocamento de algo de um lugar para outro e na informática o objeto movimentado pode ser uma aplicação ou dados. Entretanto, nesse contexto especificamente, a migração não possui o foco em mover nenhuma aplicação ou dado e sim a responsabilidade por uma tarefa. Essa migração é o mecanismo que nos permite ter uma aplicação distribuída dinamicamente.

Por tanto deve ser feito a distinção entre distribuição estática e dinâmica de uma aplicação. A distribuição estática acontece em tempo de inicialização e permite que uma

aplicação execute em vários dispositivo com um propósito comum. É possível ter a mesma aplicação dividia em 3 tarefas, citada anteriormente sem nenhum problema. Como restrição a responsabilidade pelas tarefas não poderão ser alteradas em tempo de execução.

É aqui onde entra a distribuição dinâmica, podendo ocorrer através de uma migração interativa ou automática. A migração interativa, como o próprio nome diz, requer a interação do usuário para acontecer. Um exemplo de onde isso é interessante, pode ser observado caso o usuário queira mudar o dispositivo de captura dos eventos, por exemplo. Não havendo nada diferente entre os dois dispositivos para disparar qualquer ação de migração automaticamente por meio de um algoritmo a interatividade é crucial.

A interatividade pode acontecer por exemplo com a exibição de alguma informação na tela seguido pela seleção do usuário, ou no exemplo citado, com o contato via *Near Field Communication* (NFC) do dispositivo de captura de eventos e o dispositivo de exibição. A interatividade não se limita a seleção de recursos físicos, podendo também servir para a seleção de recursos lógicos, por exemplo, através de um menu de configurações.

Já a migração automática é disparada por algum algoritmo, podendo acontecer de duas maneiras. Pode ser explícita no código, de tal forma que quando a execução atingir determinado ponto a migração ocorrerá invariavelmente. Pode ser disparada pelos algoritmos de negociação baseado nos recursos de cada dispositivo, nas regras de negócio, no funcionamento da aplicação, e de tantas outras maneiras quanto a criatividade permitir.

Como um tema para pesquisas futuras a negociação pode usar de técnicas de aprendizagem de máquina para tomar decisões como a migração, adaptação, negação ou outras.

5.3.5 ADAPTAÇÃO DE CONTEÚDO

As funcionalidades de negociação e migração de tarefas entre nuvem e cliente podem ser usadas com o propósito de aumentar a QoE, mas em alguns casos a adaptação do conteúdo é uma alternativa menos custosa ou até mesmo mais adequada. Por esse motivo o módulo de negociação pode tomar como ação a adaptação de conteúdo em vez da migração, dependendo do interesse dos *stakeholders*.

A adaptação de conteúdo pode ser feita em tempo de execução o que consome uma boa quantidade de recursos de computação ou pode ser feita *offline* que por sua vez consome muito recurso de armazenamento. A adaptação *offline* consiste em gerar diferentes versões de um conteúdo, como, imagem, áudio ou vídeo, para atender diferentes condições, tais

como, telas de tamanhos diferentes, largura de banda limitada ou carência de suporte a um determinado formato. Ver Figura 5.4 que demonstra um exemplo de aplicação de *streaming* de vídeo.

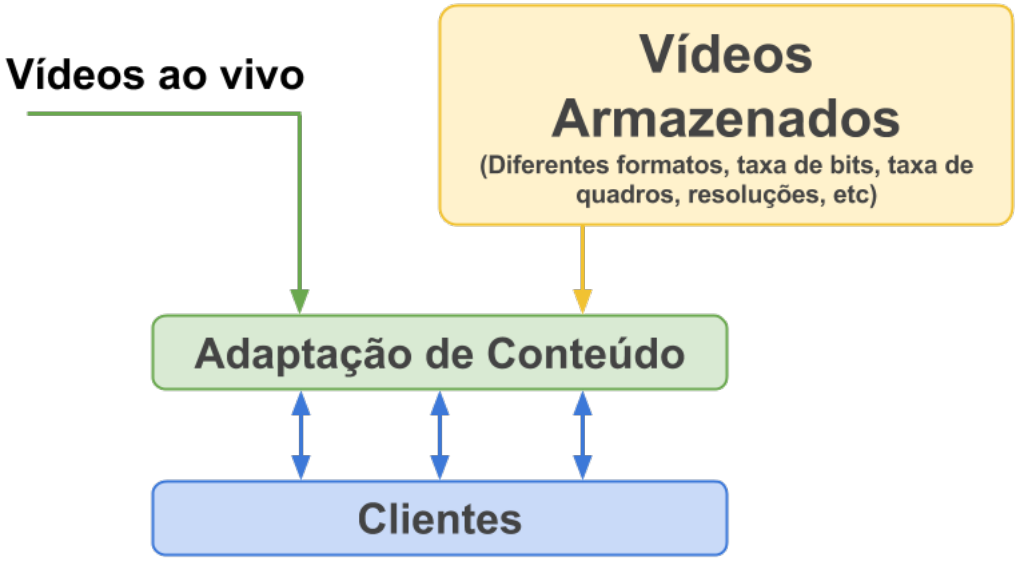

Figura 5.4: Adaptação de vídeo ao vivo e armazenado (ZHU et al., 2011)

5.3.6 GERENTE DE POLÍTICAS

Ao envolver a nuvem é necessário o uso do módulo Policy Manager para servir de interface entre as regras de negócio da nuvem e os dispositivos na rede local envolvidos na aplicação em questão.

Devido a arquitetura distribuída um nó deve ser escolhido para ser responsável por esse módulo. A escolha pode ser aleatória ou baseada em algum atributo, por exemplo, o dispositivo com maior poder de processamento.

Outra observação importante é que mais de uma aplicação pode ser executada na mesma rede e algumas vezes envolvendo dispositivos em comum, nesse caso cada grupo executando uma aplicação terá disponível seu próprio Gerente de Políticas.

5.3.7 MONITOR DE INSTÂNCIAS

Como mais de uma instância pode executar por dispositivo esse módulo se faz necessário para aumentar o poder do framework, permitindo que além de todos os outros fatores já citados, também seja levado em conta o funcionamento da aplicação.

5.3.8 REPORTE PERIÓDICO

Os dispositivos devem anunciar suas capacidades e contexto periodicamente para que outros dispositivos possam atuar de forma a otimizar a execução para dada aplicação e minimizar o impacto nos envolvidos. Não só os recursos dos dispositivos devem ser anunciados mas também as informações de como a aplicação está se comportando e as restrições políticas.

O tempo entre os anúncios deve ser pensado levando em conta dois pontos de vista, o primeiro é que quanto menor for o período entre anúncios mais rápido será a resposta a condições indesejadas, entretanto, quanto mais lento for, menor será a sobrecarga imposta na rede.

Uma última observação quanto aos agentes de coleta de informação é que quanto menos informações forem monitoradas, menor será o gasto computacional e consumo de energia, além de tornar o processo menos intrusivo no que diz respeito a privacidade do usuário.

6 MIDDLEWARE

Alguns componentes foram pensados para que o *middleware* consiga fornecer as funcionalidades desejadas, sendo os principais a tabela de fardo, a lista de associados, objetos nômades e os objetos de comunicação.

6.1 COMPONENTES DO MIDDLEWARE

A **tabela de fardo** contém as informações de quais instâncias da aplicação estão executando cada tarefa. Essa tabela deve conter o identificador da tarefa e para cada identificador uma lista de tarefas dependentes e uma lista de instâncias responsáveis pela mesma.

A **lista de dependências** pode ser tanto uma lista de tarefas que dependem da atual ou uma lista de tarefas das quais a atual depende. Essa decisão afeta apenas o algoritmo para criação dos objetos de comunicação. A **lista de instância** deve representar de maneira única uma instância da aplicação, por exemplo, pode-se usar o endereço IP (*Internet Protocol*) ou o endereço IP conciliado com a porta TCP/UDP (*Transmission Control Protocol*/*User Datagram Protocol*) para permitir várias instâncias por computador. Ver Figura 6.1.

Tabela de Fardo		
Tarefa	**Dependências**	**Instâncias**
A_TASK	B_TASK	(192.0.2.1:42013)
B_TASK	C1_TASK, C2_TASK	(192.0.2.1:42013)
C1_TASK	D_TASK	(192.0.2.2:42013)
C2_TASK	D_TASK	(192.0.2.3:42013)
D_TASK		(192.0.2.1:42013), (192.0.2.4:42013)

Figura 6.1: Tabela de Fardo

A **lista de associados** possui uma tarefa mais simples e intermediária. Ela é res-

ponsável por armazenar as informações de cada instância que se juntou a aplicação antes mesmo dessa assumir qualquer responsabilidade na cadeia de processamento. Tanto a tabela de fardo quanto a lista de associados das instâncias participantes devem estar em constante sincronismo.

Os **objetos nômades** contém as informações úteis de cada tarefa sendo processada, ou seja, sempre que uma tarefa deseja enviar informações para outra essa deve criar um objeto nômade com as informações e então enviar para a próxima tarefa. Assim, para que a comunicação entre as tarefas ocorra esses objetos são enviados de uma tarefa para a outra, o que justifica seu nome. Vale ressaltar que o escopo desses objetos está limitado a tarefa que enviou e suas dependentes.

Baseado nas tarefas que são executadas em determinada instância devem ser tomadas decisões diferentes, por exemplo, enviar o objeto nômade para a próxima tarefa na mesma instância e armazená-lo em uma fila ou criar uma *thread* para envio remoto. Esses objetos, filas e *threads*, entre outros são denominados **objetos de comunicação**.

6.2 MODULARIDADE E DEPENDÊNCIAS

Com os elementos apresentados acima, o *middleware* fornece a capacidade de criar **aplicações modulares**, onde cada módulo é referido como uma tarefa da aplicação. Essas tarefas podem ser distribuídas entre diversas instâncias da aplicação e a comunicação entre elas acontece através dos objetos nômades que é orquestrada pelo *middleware*.

Com essa distribuição de tarefas vem a necessidade de especificar as dependências umas das outras. Para tratar essa questão é proposto que as tarefas se relacionem seguindo as cardinalidades um-pra-um (1:1), um-pra-muitos (1:n) e muitos-pra-um (n:1). Ver Figura 6.2.

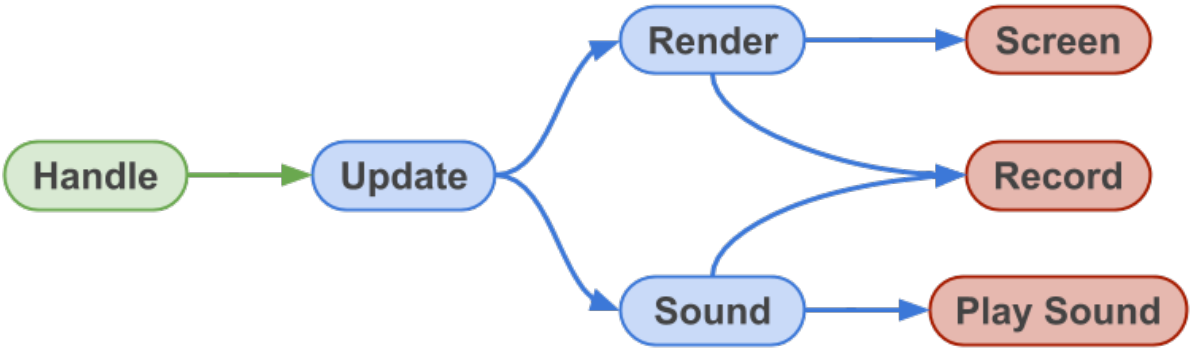

Figura 6.2: Aplicação com diferentes dependências quanto à cardinalidade

Como exemplo dessas cardinalidades pode-se ter um jogo dividido nas seguintes tarefas: manipulação dos eventos de entrada (*Handle*); atualização do estado do jogo (*Update*) que depende dos eventos; renderização da imagem (*Render*) que depende do estado do jogo; criação do *buffer* de som (*Sound*) que também depende do estado do jogo; exibição da tela (*Screen*) que depende da etapa de renderização; reprodução do som (*Play Sound*) que depende da criação do *buffer* de som; e por último uma etapa para gravar tanto a imagem quanto o som do jogo (*Record*) que dependem de ambos renderização e criação do *buffer* de som.

Esse exemplo demonstra a cardinalidade 1:1 entre as tarefas de captura dos eventos de entrada e atualização do estado do jogo. Um exemplo de cardinalidade 1:n está entre a tarefa de atualização do estado do jogo e as tarefas de renderização e criação do *buffer* de som. E a cardinalidade n:1 está presente entre essas últimas duas tarefas citadas e a tarefa de gravação.

Essas dependências têm como gatilho o fim da execução de uma tarefa e como ação o início de outra tarefa. São dependências do tipo fim-início, mas podem ser do tipo fim-fim ou início-início. Como exemplo, as tarefas de exibição da tela e reprodução do som podem ter a restrição de iniciarem juntas mantendo a sincronia entre imagem e som.

É proposto que o desenvolvedor da aplicação possa especificar esses requisitos tanto através de código quanto de maneira visual. Uma das abordagens visuais é a construção de um grafo direcional, mas esse método apresenta limitações, como a incapacidade de especificar outros tipos de dependências que não sejam fim-início. Uma segunda abordagem é o uso do diagrama de Gantt, como na Figura 6.3.

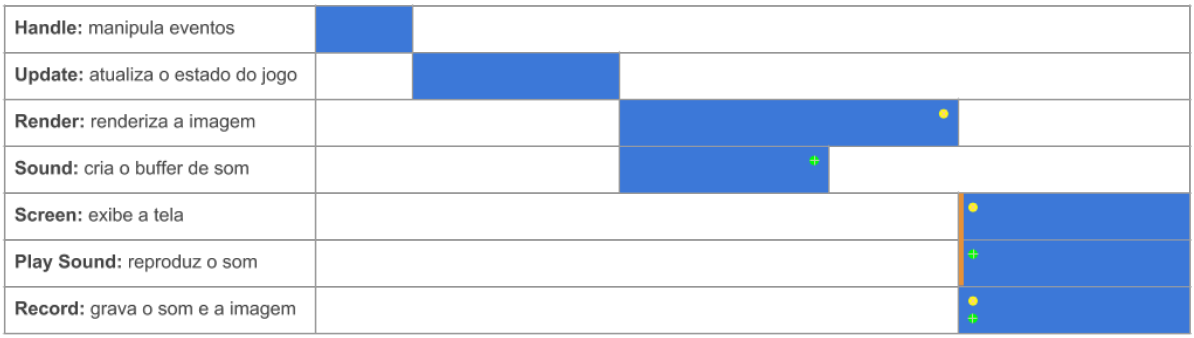

Figura 6.3: Representação das dependências com diagrama de Gantt

6.3 MÚLTIPLAS INSTÂNCIAS POR TAREFA

Múltiplas dependências abrem caminho para um caso mais específico, onde múltiplas instâncias podem executar a mesma tarefa. Pense em um cenário mais simples onde apenas 3 das tarefas anteriores estão presente, são elas, captura dos eventos de entrada, atualização do estado do jogo e renderização.

Nesse cenário podem haver múltiplos jogadores executando a primeira etapa, enviando-as para um computador processar e retornando para cada um o estado do jogo a ser renderizado. Nesse caso temos uma tarefa central com dependências n:1 e 1:n, mas as tarefas anteriores são iguais e por isso deve haver um tratamento especial para esse caso. Ver Figura 6.4.

Uma maneira de se tratar essa questão é o *middleware* fornecer um método para identificar o tráfego originado em cada instância e outro método para processar os objetos independentes da origem (i.e. por ordem de chegada).

Figura 6.4: Múltiplas instâncias responsáveis por tarefa

Nem sempre a solução para o multiprocessamento estará no *middleware*, por exemplo, no caso de um único jogador desejar que a etapa de processamento seja executada em vários computadores para aumentar o desempenho, não é necessário usar o *middleware*. Pode ser usado um *cluster* de forma transparente onde o *middleware* o enxerga como uma única instância e o próprio *cluster* se encarrega de dividir o processamento. A utilização do *middleware* para processamento distribuído possui limitações no quesito granularidade. Ver Figura 6.5.

6.4 SUPORTE A ASSINCRONISMO E SINCRONISMO

A execução de cada tarefa ou módulo é independente tanto no escopo da nuvem quanto local. Isso quer dizer que uma tarefa pode acontecer de maneira dessincronizada da outra, por exemplo, a tarefa de captura dos eventos de entrada pode acontecer 60 vezes

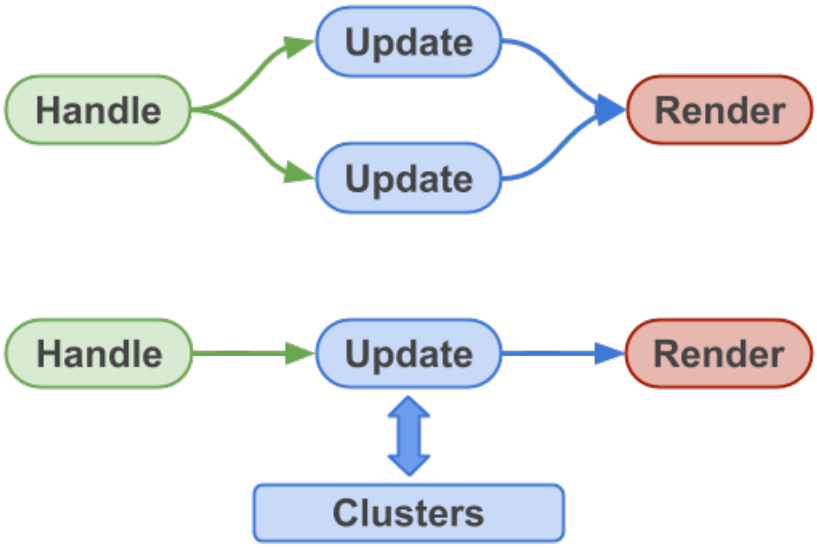

Figura 6.5: Diferença entre múltiplos envolvidos em uma tarefa e uso de cluster

por segundo, mas a tarefa de processamento do estado da aplicação pode acontecer apenas 30 vezes por segundo.

O desenvolvedor terá apenas que decidir como ele quer tratar o envio de dados entre um módulo e outro. Nesse caso apresentado, o desenvolvedor pode desejar que nenhum evento capturado seja perdido, então no módulo responsável por processar o estado da aplicação, os dados recebidos devem ser enfileirados. Nesse momento é oportuno fazer duas observações. A primeira é que para o desenvolvedor é transparente se as duas tarefas estão sendo executadas local ou remotamente, essa responsabilidade é do *middleware*.

A segunda observação é que no caso de uma tarefa ser executada a uma taxa mais elevada que sua sucessora e usar o enfileiramento para garantir o processamento de todos os dados, implica que o tamanho da fila vai crescer indiscriminadamente. O uso do enfileiramento é interessante para anomalias no funcionamento que façam a taxa variar momentaneamente e não para atraso contínuo.

Outro uso interessante pode ser visto no seguinte cenário. A tarefa de captura dos eventos de entrada é executada a uma taxa de 60 vezes por segundo e as tarefas de processamento do estado da aplicação e também exibição do resultado são executadas em outro dispositivo a 30 vezes por segundo.

Assumindo que a taxa limitada tem por objetivo principal a exibição de um vídeo a 30 FPS, então a cada vez que a tarefa de processamento do estado da aplicação for executada ela pode entrar em um *loop* até esvaziar a fila e então seguir para a exibição.

Em resumo, a cada dois pacotes processados na segunda tarefa, apenas um quadro do vídeo será exibido, porém, com as informações atualizadas.

É importante que ao tentar esvaziar a fila, essa seja bloqueada para que nenhum novo item possa ser adicionado ou pode acontecer o fenômeno de *starvation*. Para tratar esse problema, um método para retornar uma lista com todos os elementos atualmente na fila pode ser fornecido para o desenvolvedor através da API do *middleware*.

Como alternativa a garantia de processamento é possível que cada objeto enviado substitua o anterior. Isso é útil para evitar o problema de estouro de fila quando o atraso é permanente e não há a necessidade de garantia de processamento. Assumindo o cenário onde todos os eventos são processados mantendo o estado da aplicação atualizado a uma taxa de 60 vezes por segundo e que a exibição é processada a uma taxa de 30 vezes por segundo, é aceitável a perda de informações de exibição uma vez que é improvável que nossos olhos percebam qualquer ganho significativo em uma taxa maior do que essa.

Até o momento foi falado sobre dessincronismo, mas o sincronismo também pode ser interessante em determinadas aplicações. Como já foi dito sobre a Figura 6.2, as tarefas de exibição da tela e reprodução do som são baseadas no mesmo estado da aplicação e devem ser reproduzidas em sincronia mesmo estando em dispositivos diferentes. Nesse caso um mecanismo deve ser provido pelo *middleware* a fim de garantir que tal restrição seja atendida.

6.5 MANIPULAÇÃO DE TOPOLOGIA

O *middleware* também é intencionado a fornecer métodos para se juntar a outra instância da aplicação ou grupo de instâncias, bem como deixar o grupo ou alterar as responsabilidades dentro do mesmo.

Esses são os mecanismos mais simples, outros mais avançados têm o propósito de aumentar o nível de abstração. Alguns deles são o mecanismo de descoberta que procura por outras instâncias da aplicação dentro da rede local, o mecanismo de *heartbeat* que verifica se algum problema aconteceu com os participantes e a mecanismo de negociação que como visto anteriormente pode trabalhar em vários níveis de abstração.

O *middleware* deve ter também uma interface dedicada a tratar alguns problemas pertinentes a comunicação sobre a Internet, onde os dados trafegarão em redes além do domínio do usuário. Questões como autenticação e tradução de endereços de escopo local

são algumas das responsabilidades dessa interface.

Um exemplo de problema a se tratar é o *firewall*, que normalmente não interfere na comunicação interna, permitindo que cada instância se comunique diretamente umas com as outras. Porém ao estender a comunicação para fora da rede local a instância externa pode não ter acesso direto ao dispositivo de destino devido a uma restrição no *firewall*. Sendo assim, o caminho de origem a única maneira de se retornar informações para a rede local, mesmo que a instância de origem não seja a mesma de destino. Esse e outros problemas devem ser tratados por essa interface. Ver Figura 6.6.

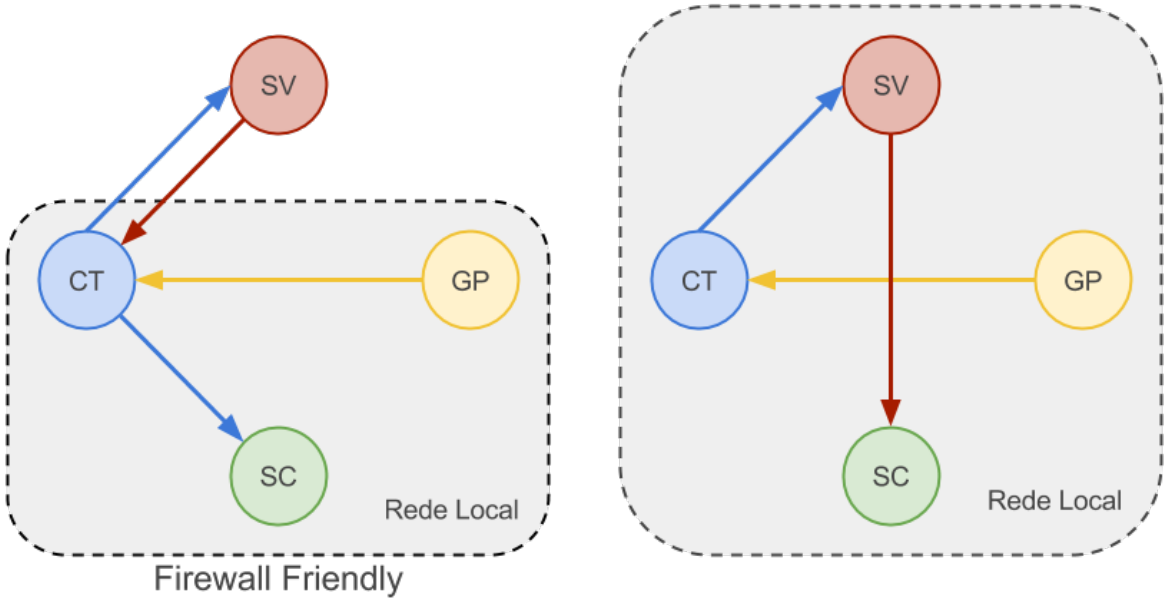

Figura 6.6: Diferença entre aplicação executando em ambiente local e ambiente externo firewall-friendly

6.6 DEFINIÇÃO DE APIS UNIFORMES

É proposto que os componentes da arquitetura de nuvem multimídia tenham uma API clara e padronizada seguindo boas práticas da indústria. A API no *middleware* visa fornecer comunicação entre os componentes da arquitetura e facilidades no desenvolvimento, por exemplo, distribuição e paralelismo, migração de responsabilidades, sincronização, segurança, uso transparente da infraestrutura em nuvem e QaaS, múltiplos usuários, desenvolvimento em tempo de execução, migração de código, etc.

A API do MEC conciliada a sua criação modular permite que o administrador possa escolher o que melhor se alinha com a empresa, por exemplo, o *cluster* que será usado, o *load balancer* ou o algoritmo de negociação. É possível criar um *front-end* de gerência alternativo usando essa API.

A API do MSP possui as mesmas características da API anterior sendo intencionado para provedores de conteúdo que desejam entregar serviços de multimídia com alta QoE para seus clientes usando MECs e o *middleware* proposto. O administrador, pode, por exemplo, adicionar e remover MECs e políticas que serão usadas pelo *middleware*. É possível criar um *front-end* de gerência alternativo usando essa API.

A API do MSP também permite a criação de um *front-end* específico para o serviço prestado, por exemplo, empresas que fornecem um serviço *Web* de *streaming* de vídeo, como YouTube, DailyMotion ou Twitch, podem usufruir dos MECs com a criação de suas páginas em cima dessa arquitetura.

7 PROVA DE CONCEITO

O *framework* proposto pode ser considerado um trabalho de longo prazo pelos diversos aspectos a serem abordados, conforme notado nas discussões anteriores. Questões como os algoritmos de negociação, a estrutura de especificação de requisitos e recursos, dentre outros detalhes são aqui definidos apenas de forma preliminar, para viabilizar o desenvolvimento de uma prova de conceito nesta dissertação.

Até o momento, o *middleware* é onde estão sendo direcionados os esforços, uma vez que esse é a base de todas as aplicações. Acredita-se que ele sirva como prova de conceito ainda que sem a implementação completa de MEC e MSP, pois esses dois adicionam principalmente negociação em nível de política e o mecanismo para isso não será muito diferente das demais negociações incluídas no cenário da prova de conceito. Uma vez que o caminho entre MEC e cliente esteja estabelecido, ou seja, endereços de mesmo escopo, sem restrições de *firewall* e afins, a comunicação será idêntica a feita localmente.

A linguagem escolhida para o desenvolvimento foi Python[10]. A decisão por essa linguagem se deve principalmente por sua simplicidade, padronização, documentação e pela comunidade ativa, que sempre se preocupa em garantir que não só a linguagem sigam esses requisitos, mas também os códigos escritos nela. Algumas características técnicas também a tornam atrativa para criação de um *middleware*, como o suporte a introspecção, reificação e reflexão.

O protótipo usado como prova de conceito foi desenvolvido sobre o sistema operacional GNU/Linux, mas por ser uma linguagem multiplataforma e com muitos módulos também multiplataforma, é simples portá-lo para outro sistema operacional. Por ser uma linguagem interpretada é possível migrar o código entre dispositivos e simplesmente executá-los.

O *middleware* atualmente é desenvolvido como sendo um módulo para o Python o que restringe seu funcionamento à aplicações usando a mesma linguagem. Por comodidade é usado o serializador pickle que gera um formato binário dos objetos, o que se torna outro fator limitante para aplicações em outras linguagens. Para resolver essa restrição de linguagem espera-se no futuro criar uma interface bem definida para a serialização de objetos e transformar o *middleware* em um serviço do sistema operacional.

[10]https://www.python.org/

7.1 USO DO MIDDLEWARE

Para se usar o *middleware*, deve ser criado um objeto que herde a classe `Middleware`, onde cada método pode ser a implementação de uma tarefa da aplicação, mas não necessariamente todo método precisa ser uma tarefa. Dentro do arquivo de *setting* no *middleware*, as tarefas devem ser especificadas seguido das tarefas que dependem dela, como no exemplo abaixo baseado na aplicação da Figura 6.2.

```
HANDLE_TASK = 0
UPDATE_TASK = 1
RENDER_TASK = 2
SOUND_TASK = 3
SCREEN_TASK = 4
PLAY_SOUND_TASK = 5
RECORD_TASK = 6

ALL_TASK = (
    (HANDLE_TASK, [UPDATE_TASK]),
    (UPDATE_TASK, [RENDER_TASK, SOUND_TASK]),
    (RENDER_TASK, [SCREEN_TASK, RECORD_TASK]),
    (SOUND_TASK, [PLAY_SOUND_TASK, RECORD_TASK]),
    (SCREEN_TASK, []),
    (PLAY_SOUND_TASK, []),
    (RECORD_TASK, [])
)
```

Dentro da implementação das tarefas devem ser usadas as funções para recuperar e enviar os **objetos nômades**. Para receber dados deve-se chamar a função especificando a tarefa de origem e de destino, isso devido ao funcionamento interno do *middleware* para garantir suporte à múltiplas dependências e múltiplos responsáveis pela mesma tarefa. Já para enviar deve-se chamar a função especificando apenas qual é a tarefa que está enviando para que o *middleware* consiga realizar a entrega adequadamente.

Existem basicamente dois métodos para enviar e dois métodos para receber dados, um onde todas as instâncias de uma mesma tarefa serão envolvidas (`fetch_all` e `update_all`)

e outro onde se pode informar o endereço de uma instância específica (`fetch` e `update`).

```
fetch      (prev_task, task, src, method=FIRST_OBJ)
fetch_all  (prev_task, task, method=FIRST_OBJ)
update     (task, obj, dst, queue=True)
update_all (task, obj, queue=True)
```

Os métodos de `fetch` suportam 3 modos de operação, um onde retorna o objeto mais antigo da fila (`FIRST_OBJ`), um onde retorna o objeto mais recente da fila descartando os antigos (`LAST_OBJ`) e por último um que retorna todos os objetos da fila (`ALL_OBJ`). Ver Figura 7.1.

Os métodos de `update` suportam 2 modos de operação, um onde o objeto enviado será enfileirado e outro onde apenas o último objeto enviado estará disponível para a tarefa de destino. Essa diferenciação é útil para tarefas que não necessitam de garantia de entrega. O objeto a ser enviado também deve ser passado como parâmetro. Ver exemplo abaixo.

```
def game_update(self):
    instance_list = self.fetch_all(setting.HANDLE_TASK,
                                   setting.UPDATE_TASK,
                                   middleware.ALL_OBJ)
    # implementação da tarefa ...
    self.update_all(setting.UPDATE_TASK, self.game_state, False)
```

Para não ter que especificar a tarefa atual pode-se criar os objetos nômades a partir da classe `NomadObject`. As funções apresentadas aqui funcionam como um *proxy*, elas simplificam o desenvolvimento delegando para o *middleware* a responsabilidade de descobrir as instâncias envolvidas em cada tarefa.

7.2 MANIPULAÇÃO DA TOPOLOGIA

Ao iniciar a aplicação a **tabela de fardo** é inicializada com o endereço da própria instância para todas as tarefas e os **objetos de comunicação** são criados, como filas para receber os objetos nômades e *threads* para enviá-los a instâncias remotas (na inicialização *threads* de envio não são necessárias). Os objetos de comunicação são reconstruídos sempre que

há alteração na topologia, por exemplo, quando uma nova instância se junta ao grupo e se responsabiliza por alguma tarefa.

Vale observar que apenas se juntar ao grupo não altera a topologia, a instância em questão apenas é adicionada à **lista de associados** das demais. Para alterar a topologia as funções mais básicas são `join`, `migrate` e `leave`.

```
join    (remote_addr, remote_port=42013)
migrate (task, method, addr=None, port=42013)
leave   ()
```

A função `join` espera como argumento o endereço de uma instância do grupo e como resposta a instância remota envia a tabela de fardo do grupo e notifica as demais instâncias da alteração na topologia. Observe que qualquer membro do grupo pode ser escolhido para ser alvo do `join`, isso pode ser feito, por exemplo, com o primeiro endereço que responder a uma requisição de descoberta.

A função `leave` simplesmente notifica os demais membros do grupo que uma instância está saindo e depois remove todas referências a outras instâncias de sua própria tabela de fardo e lista de associados.

A função `migrate` recebe como parâmetro a tarefa que deseja alterar a responsabilidade, qual o tipo de alteração deseja fazer e opcionalmente qual o endereço para alteração. Caso o endereço não seja especificado se assume o endereço da instância que executa a chamada. Atualmente 3 tipos de operações são suportadas, trocar o responsável pela tarefa (`SET_TASK`), adicionar mais um responsável pela tarefa (`ADD_TO_TASK`) e remover um dos responsáveis pela tarefa (`DEL_FROM_TASK`). Essas funções podem ser chamadas de acordo com a lógica da aplicação ou por camadas de abstração do próprio *middleware* como será explicado a seguir. Ver exemplo de uso na aplicação abaixo.

```
def game_migration(self):
    if self.local_event.lshift_key.pressed:
        if self.local_event.f5_key.pressed:
            self.leave()
        if self.local_event.f6_key.pressed:
            self.migrate(setting.HANDLE_TASK, middleware.DEL_MIGRATE)
        if self.local_event.f7_key.pressed:
```

```
            self.migrate(setting.UPDATE_TASK, middleware.SET_MIGRATE)
        if self.local_event.f8_key.pressed:
            self.migrate(setting.RENDER_TASK, middleware.DEL_MIGRATE)
    else:
        if self.local_event.f5_key.pressed:
            self.join(setting.TEST_IP, setting.TEST_PORT)
        if self.local_event.f6_key.pressed:
            self.migrate(setting.HANDLE_TASK, middleware.ADD_MIGRATE)
        if self.local_event.f7_key.pressed:
            self.migrate(setting.UPDATE_TASK, middleware.SET_MIGRATE)
        if self.local_event.f8_key.pressed:
            self.migrate(setting.RENDER_TASK, middleware.ADD_MIGRATE)
```

Com essas funcionalidades básicas implementadas, camadas de abstração podem ser desenvolvidas em cima dela: (i) descobrir e se juntar a um grupo; (ii) *heartbeat* para não necessitar que uma instância explicitamente saia do grupo; ou (ii) um mecanismo que ao detectar que alguma tarefa ficou sem responsável automaticamente encontre o melhor substituto.

7.3 OBJETOS DE COMUNICAÇÃO

Agora será falado dos objetos de comunicação e como eles foram pensados para aumentar a flexibilidade do *middleware* e ganhar em desempenho. Essa seção descreve funcionalidades que não são visíveis para o desenvolvedor e por tanto algumas alterações podem ser feitas sem impactar no desenvolvimento.

`MultiQueue` é um dos objetos de comunicação, ela é responsável por gerenciar o acesso das tarefas aos objetos recebidos. O seguinte exemplo demonstra seu propósito, supondo que a tarefa executada em uma instância deve enviar os objetos gerados para outras 3 tarefas, as quais estão todas em uma segunda instância.

Se for apenas uma fila, cada tarefa dependente idealmente receberá apenas um terço dos objetos. A abordagem mais simples para resolver isso é ter uma fila compartilhada entre a primeira tarefa e cada uma das tarefas dependentes, resultando em 3 filas e talvez até no envio repetido dos objetos para cada uma dessas filas.

Uma solução inicial para esse problema é associar um contador a cada objeto enfileirado com o valor igual ao número de tarefas, e então cada vez que uma tarefa requisitar o próximo objeto, o contador é decrementado. Quando todas as tarefas tiverem requisitado, o contador atinge o valor 0 e o objeto é removido.

O problema dessa solução é que as tarefas são assíncronas e podem consumir os objetos em tempos diferentes. Para resolver isso adicionamos à solução anterior um identificador (*identifier* — ID) para cada objeto e monitoramos qual objeto cada tarefa já requisitou. O contador serve apenas para remover o objeto da fila quando todas as tarefas já o tiverem pego. Ver Figura 7.1.

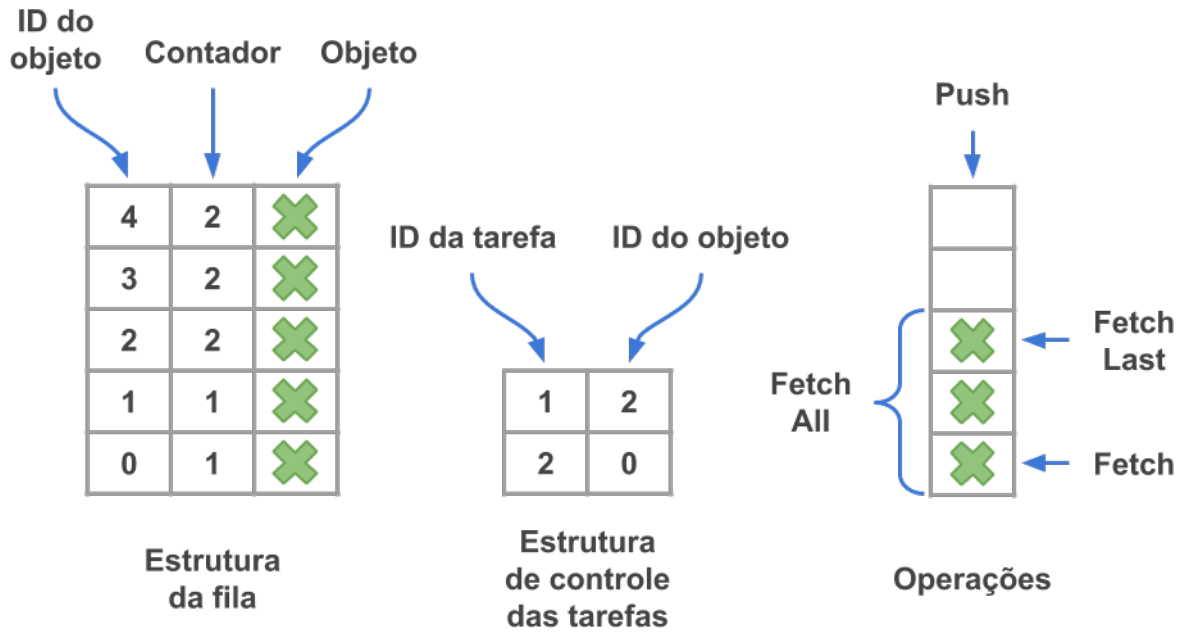

Figura 7.1: Estrutura interna da MultiQueue

Cada instância verifica tarefa por tarefa para identificar suas responsabilidades. Para cada tarefa é verificado quais tarefas dependentes dessa são executadas pela própria instância e então é criado uma `MultiQueue` com contador igual à quantidade de tarefas. Nesse mesmo momento é avaliado se a tarefa em questão é de responsabilidade da instância, se sim, é criado uma *thread* de envio para as tarefas dependentes que não são. Ver Figura 7.2.

Outro objeto importante no processo de comunicação que não necessariamente é um objeto de comunicação é o `MultiLock`. A tabela de fardo pode alterar a qualquer momento e é necessário que ao executar as alterações nos objetos de comunicação, como criar e destruir, nenhuma tarefa o esteja usando.

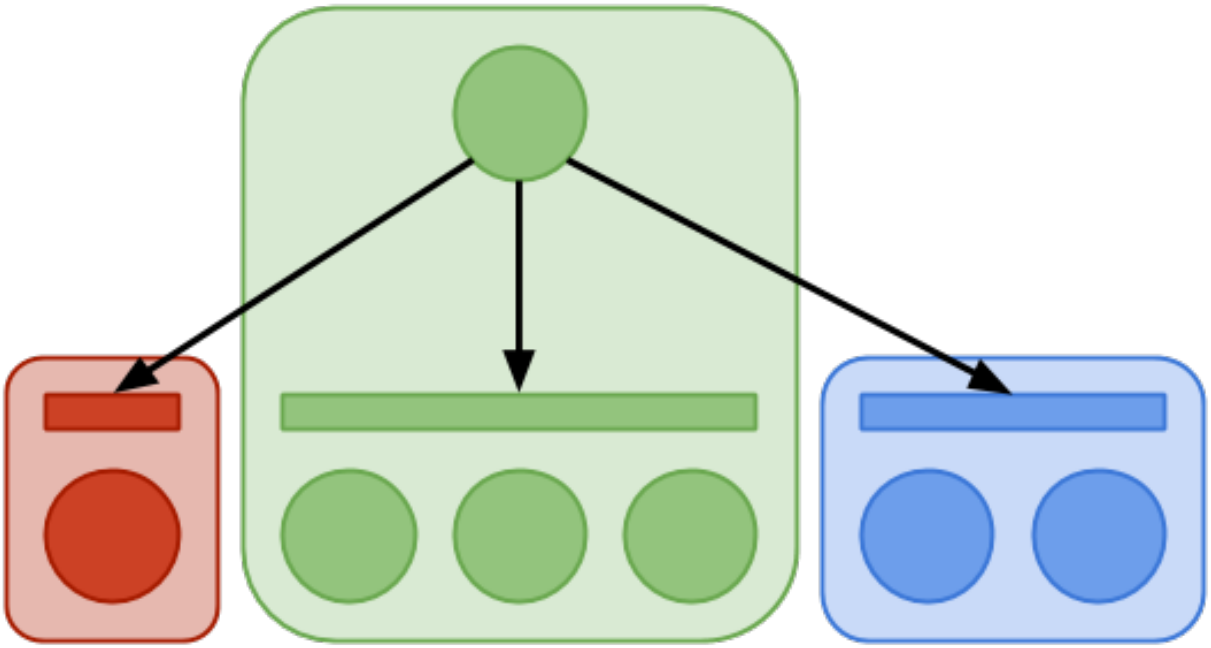

Figura 7.2: Processo de criação dos objetos de comunicação

Para isso a abordagem mais simples é usar um objeto que forneça o mecanismo de *mutual exclusion* (mutex) sempre que qualquer tarefa for usar um dos objetos de comunicação e sempre que a tabela de fardo for alterada. Essa solução possui um problema de desempenho, pois as tarefas dependerão umas das outras pra entrarem em sua região crítica sendo que os objetos usados por cada uma delas são diferentes. Para solucionar isso foi criado o `MultiLock` onde cada tarefa bloqueia apenas os objetos relacionados a si mesma, enquanto a alteração da tabela de fardo bloqueia todos os objetos de comunicação.

7.4 APLICAÇÃO

É complicado desenvolver um *middleware* sem uma aplicação para testá-lo, com esse propósito um jogo está sendo criado em paralelo com o *middleware*. Em contrapartida o desenvolvimento do jogo dentre muitas outras possíveis aplicações é uma das que gera maior demanda para a evolução do *middleware*.

O jogo pode fazer uso do *cluster* de armazenamento para salvar informações do usuário, bem como do *cluster* de CPUs para o processamento do estado do jogo e do *cluster* de GPUs para o processamento de vídeo, necessita garantir a sincronização de áudio e vídeo mesmo quando executando em dispositivos diferentes e além da necessidade de processamento o jogo traz restrições de tempo devido à interatividade.

A pergunta que deve ser feita neste momento é “O que o jogo ganha por ser desenvolvido em cima do *middleware*?”. O primeiro ponto que deve ser destacado é o desempenho, o qual já pode ser sentido, por exemplo, ao delegar a etapa de renderização da imagem para um computador com maior poder de processamento.

Graças ao *middleware* o jogo também pode ter suas tarefas executadas de maneira distribuída entre diferentes tipos de dispositivo, por exemplo, a captura de *inputs* e o estado do jogo podem ser processados em um *smart gamepad* enquanto a imagem é exibida em uma TV. Isso permite inclusive trocar de TV ou de cômodo e continuar o jogo exatamente onde estava.

7.5 PROTÓTIPO DO JOGO

Nessa seção será feito um resumo do estado em que se encontra o jogo e onde espera-se chegar. A Figura 7.3 mostra a tela do jogo em execução.

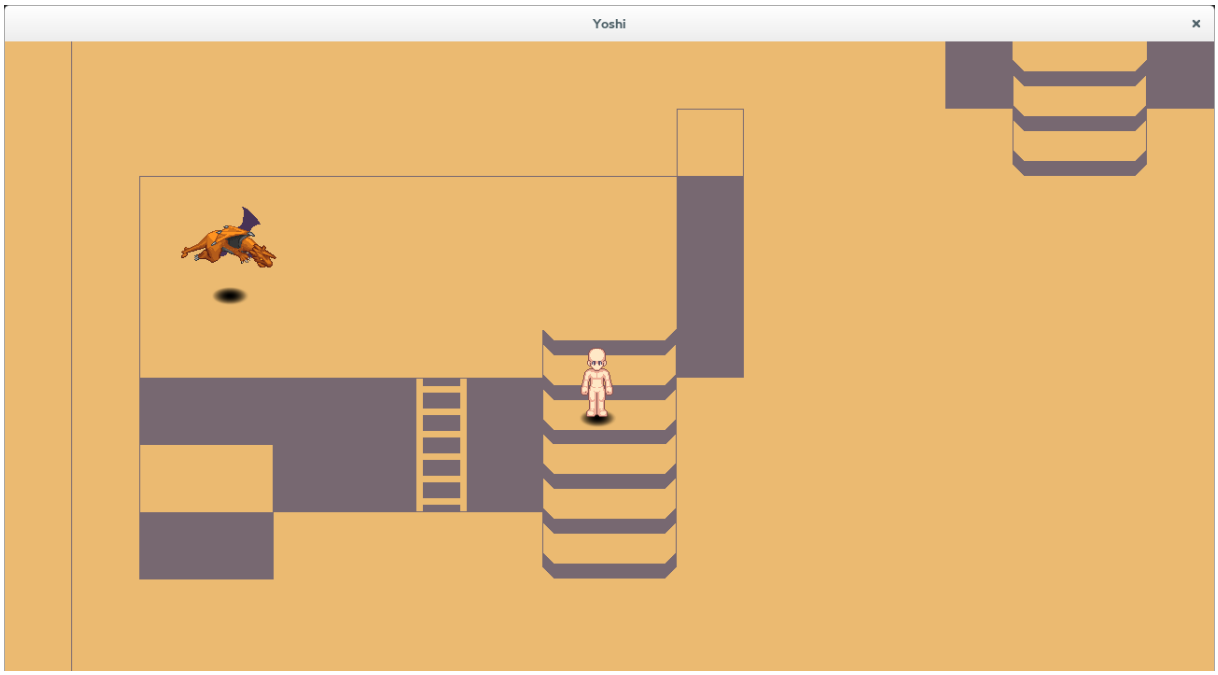

Figura 7.3: Tela do protótipo do jogo em execução

O jogo é composto por 3 tarefas, como nos exemplos já citados anteriormente, captura de eventos, processamento do estado da aplicação e renderização. O propósito é atingir a estrutura referida na Figura 6.2 levando a necessidade de implementar os mecanismos relacionados a sincronização de tarefas em dispositivos remotos e adaptação de conteúdo.

Ainda quanto à estrutura, o jogo é multiusuário permitindo múltiplos envolvidos nas

tarefas de manipulação de eventos e renderização.

No atual estado do jogo todo o processo de manipulação da topologia é feito de maneira interativa baseado em eventos do teclado. O `join` é executado através da tecla F5 em vez do uso de descoberta e união automática. O `leave` é executado através da combinação de teclas SHIFT+F5 em vez do uso do mecanismo de *heartbeat*. A alteração de responsabilidade é feita através das teclas F6 à F8, onde a responsabilidade de cada etapa pode ser tomada para a própria instância ao pressionar essas teclas ou delegada para outra instância por pressionar as teclas combinadas com a tecla SHIFT.

O mecanismo de descoberta ainda não foi implementado, por isso os identificadores das instâncias remotas devem estar explícitos no código do jogo ou em algum arquivo de configuração.

8 CONSIDERAÇÕES ADICIONAIS

8.1 LATÊNCIA

Temos basicamente três conceitos de atraso que podem impactar em diferentes tipos de aplicações. A latência, na prática (CHESHIRE, 1996), é o tempo de resposta de um pacote na rede e quanto maior, pior a experiência com aplicações interativas. O *jitter* é a variação entre o tempo de entrega dos dados e aplicações de *streaming*, por exemplo, áudio ou vídeo, são as que mais sofrem com o aumento dessa variação. O *skew* é utilizado para medir a diferença entre os tempos de chegada de diferentes mídias que deveriam estar sincronizadas, ou seja, a baixa qualidade nesse parâmetro impacta negativamente em aplicações multimídia que são altamente dependentes de sincronismo.

Esse trabalho tem como um de seus objetivos resolver os problemas de atraso, mas algumas considerações devem ser feitas. Mesmo trazendo o processamento para a borda da nuvem a latência pode ser um limitador de aplicações. Quando falamos de borda não necessariamente estamos falando de um único salto, muitas vezes o *Internet Service Provider* (ISP) possui estações espalhadas em uma granularidade grande de mais pra justificar a implantação de um MEC em cada uma, levando-o a colocar em um ponto central a alguns saltos do cliente.

O atraso no tráfego de dados tem vários fatores que devem ser analisados, por exemplo, o tempo para colocar os dados na rede, o tempo de transmissão em determinado meio (*link*), tempo de processamento nos nós intermediários, tempo em *buffers*, etc. A questão é encontrar qual desses fatores mais geram atrasos, por exemplo, os dados transmitidos em cabos metálicos viajam a aproximadamente 66% da velocidade da luz e a substituição desse meio pro outro com maior velocidade não surtirá um grande efeito positivo levando em consideração distâncias pequenas e as limitações físicas.

As *middle-boxes* são onde a maior parte da latência é inserida, seja no processamento ou o tempo gasto no *buffer*, e muitas vezes elas fazem mais do que apenas encaminhamento, por exemplo, filtro de pacotes. A fim de melhorar questões não só de segurança, mas também de gerenciamento e até mesmo eficiência energética, mais *overhead* é inserido na rede e como consequência isso aumento a latência.

Um exemplo de preocupação com eficiência energética que pode aumentar a latência está nos dispositivos móveis. Esses usam uma técnica para poupar energia que consiste de ligar o *transceiver* apenas por curtos períodos para receber pacotes que estão no *buffer* da estação base, técnica essa que também aumenta a latência.

Suponha que mesmo com todos esses atrasos que são inseridos em uma comunicação fora da rede local a latência seja de 20 ms. Se o desenvolvedor espera que a aplicação execute a 30 FPS, resta para a aplicação apenas 13 ms para executar todo o processamento. Esse tempo pode ser ainda pior dependendo do atraso da rede.

Supondo agora que a aplicação leve 26 ms sendo processada por uma determinada máquina, isso torna a aplicação inviável? A resposta é não, ela pode ser executada por uma máquina com mais capacidade de processamento ou por múltiplas máquinas.

Entretanto, o equívoco que esse exemplo visa evitar é pensar que ao dobrar o número de máquinas o tempo da aplicação cairá pela metade. Isso não acontece na prática, onde o ganho pode ser bem menor do que o esperado, por exemplo, 1,6x ao dobrar o número de máquinas ou 4x ao colocar 10 máquinas. Observe que esses valores são meramente para demonstração e podem variar dependendo da aplicação e do ambiente em questão, mas é importante ter em mente que ao adicionar mais máquinas o ganho tende a diminuir.

Em resumo a latência é um grande limitador nos dias de hoje para aplicações interativas e multimídia e quanto menor o atraso inserido pelo ambiente, incluindo o próprio *middleware* proposto, maior a chance de aplicações de sucesso serem criadas.

8.2 SEGURANÇA

Muitas questões de segurança devem ser pensadas para tornar o *middleware* proposto uma ferramenta viável. Do ponto de vista dos provedores deve-se garantir que as regras de negócio sejam respeitadas por cada um dos atores, que os usuários sejam submetidos a mecanismos de autenticação e autorização ao envolverem a nuvem e muitos outros pontos tradicionais de segurança ao usar serviços de terceiros. Do ponto de vista do desenvolvedor, assim como dos provedores, suas restrições devem ser garantidas, para que não sejam prejudicados.

Do ponto de vista do usuário, sua privacidade e a segurança de seus dados devem ser garantidas. Devido ao caráter adaptativo do *middleware*, algumas informações são coletadas e devem ser tratadas como dados sensíveis. Esse é um assunto delicado se levar

em consideração o envolvimento de muitos personagens, como, usuário, desenvolvedor, provedor de rede, provedor de nuvem, provedor de serviço, etc. É crucial que o mínimo de informações sejam dadas a cada um deles, a fim de evitar o mau uso.

Para aumentar a segurança das informações do usuário, mecanismos como *Multi-Factor Authetication* (MFA) devem ser fornecidos. Mecanismos para aumentar sua comodidade também podem ser usados, como *Single Sign-On* (SSO) ou *Identity-as-a-Service* (IDaaS).

Para aumentar a privacidade do usuário deve-se essencialmente diminuir a abrangência dos seus dados/informações e evitar a centralização dos mesmos. Atualmente essas questões estão ganhando muito mais visibilidade, e além das técnicas atuais, como uso de rede federada, novas técnicas estão sendo desenvolvidas, como a encriptação homomórfica (NSA-STAFF, 2014).

Por um lado a nuvem traz algumas dificuldades em relação as questões de segurança, mas por outro ela pode ser a solução. Mais especificamente em jogos, diversas técnicas são criadas para burlar o funcionamento normal (*cheats*) a fim de tirar ganhar vantagens. Enquanto isso era apenas uma brincadeira não havia problemas, entretanto agora os jogos movimentam um mercado bilionário e problemas de segurança implicam perda de dinheiro.

Uma das ferramentas para burlar jogos são os *aimbot* usados em jogos de primeira pessoa. Algumas das construções mais comuns são baseadas em análise de memória, análise de protocolo e *hook* de APIs.

A análise de memória se baseia na premissa de que os clientes devem saber o posicionamento de todos os objetos e personagens para a correta renderização. Com essas informações na memória é possível fazer uso para criação de *cheats* que identificam a coordenada de cada jogador e enviam comandos como resposta. Em resumo mirar e atirar antes mesmo de ver o personagem.

A análise do protocolo de comunicação na rede permite a criação do mesmo tipo de ferramenta, porém não precisa executar na mesma máquina que o jogo. Isso dificulta a criação de *anticheats* uma vez que eles se baseiam em analisar a máquina onde está o jogo.

O *wallhack* é um *cheat* que faz um *Dynamic-Link Library* (DLL) *Injection* no jogo para fazer o *hook* das chamadas da API do DirectX. O jogo calcula o cenário com o posicionamento dos objetos e personagens e passa essa informação para o DirectX enviar à placa de vídeo que, por fim, será visualizada em tela pelo jogador.

Nesse caso o DLL Injection tem o propósito de se instalar no meio do caminho entre o jogo e o DirectX onde alterará as solicitações que são feitas para esse último. Isso permite ganhar vantagens no jogo, por exemplo, substituir paredes por vidros transparentes ou mesmo removê-las para ver outros jogadores livremente através do mapa, pode-se mudar a iluminação para tornar claro locais que deveriam ser escuros, etc.

Outro problema para os jogos são as emulações tanto da aplicação cliente quanto da aplicação servidor. A emulação da aplicação cliente permite a criação de *bots* de automação onde os personagens executam o *farming* automaticamente, ou seja, fica coletando itens, ganhando experiência, pontos de habilidades e dinheiro. Esse tipo de ação desleal pode modificar a economia interna do jogo, fomentar mercados paralelos e até mesmo desbalancear o jogo.

Quanto a emulação da aplicação servidor, ela impacta diretamente as *publishers* que compram os direitos do jogo em determinada região, uma vez que servidores paralelos desviam o fluxo de jogadores.

Todos esses problemas são explanados em Lima et al. (2015). Mas onde a nuvem entra nisso tudo? Como pode ser observado, a maior parte desses problemas de segurança remontam ao antigo problema da falta de controle no cliente. A solução para a maior parte deles é remover o processamento dos clientes inibindo muitas dessas técnicas e dificultando a engenharia reversa de outras. E essa tarefa pode ser feita usando a arquitetura proposta nesse trabalho.

8.3 ECOSSISTEMA

Ao desenvolver um *middleware* não apenas características técnicas devem ser levadas em consideração mas todo o ecossistema. Um *middleware* não é nada sem aplicações que sustentem sua existência. Existe uma relação simbiótica entre o *middleware* e as aplicações, quanto mais aplicações, mais popular ele se torna e com mais popularidade mais aplicações serão criadas. Com o aumento de aplicações, novas demandas aparecem, problemas são relatados e a qualidade do *middleware* tende a aumentar levando novamente a estabilização da ferramenta no mercado.

Nesse caso em específico não só *middleware* e aplicações devem crescer, mas também provedores de infraestrutura e serviço devem surgir. Outro ponto importante é a documentação que deve ser o mais completa e simples possível e para isso exemplos didáticos

e práticos devem acompanhá-la. Portanto é proposto um MSP genérico na tentativa de viabilizar o uso de aplicações distribuídas entre cliente e nuvem.

É proposto também um jogo e uma aplicação de telemetria para fomentar o uso do *middleware*. Ambos projetos estão em fase inicial, bem como o desenvolvimento do próprio *middleware* que está crescendo junto às aplicações.

Quanto ao MSP é importante ressaltar que nem todos que possuem interesse em criar aplicações sobre essa arquitetura têm acesso a uma infraestrutura para executar o MEC ou algum meio de divulgar sua aplicação, limitando-a a executar apenas na rede local do usuário. Por isso sugerimos um serviço que se resume em um *front-end* para que o usuário/desenvolvedor de uma aplicação tenha acesso a uma infraestrutura.

Nesse serviço uma interface *Web* permite que qualquer desenvolvedor possa se cadastrar e fazer *upload* de uma aplicação que utilize o *middleware*, bem como, os interessados nessas aplicações podem se cadastrar para usá-las. Essa interface pode usar conceitos de rede social e *gamification* para aumentar a imersão. Baseado no plano contratado pelo usuário determinado nível de QoS será fornecido para a execução das aplicações.

O desenvolvedor da aplicação pode criar algoritmos de negociação para implementar restrições para suas aplicações, como permitir que o usuário baixe a aplicação, use localmente e envolva a nuvem quando necessário, ou pode restringir o uso da aplicação como um serviço.

Para comodidade do desenvolvedor esse serviço *Web* deve conter uma área dedicada a documentação e exemplos, além de estar disponível o *middleware* para download.

Essa é nossa proposta para um MSP genérico a fim de divulgar e facilitar o acesso ao *middleware* e projetos usando o mesmo. Entretanto, nada impede que outros provedores de serviço criem suas próprias interfaces alinhadas com seus interesses.

Espera-se que em um futuro breve, aplicações baseadas em *Web*, dispositivos móveis, envolvendo renderização de objetos 3D, computação científica intensiva além de novas ideias sejam criadas para gerar as novas demandas que são desejadas para melhorar o *middleware*.

9 TRABALHOS FUTUROS

Como trabalhos futuros é necessário o desenvolvimento do MEC e MSP. Com esses dois componentes em funcionamento será possível seguir com as pesquisas sobre a integração com *clusters* existentes no mercado, técnicas para prover QoS e mobilidade entre MECs e mesmo CSPs. Esses dois componentes também reforçam a importância de se investir tempo em questões de segurança.

Quanto ao *middleware*, ainda devem ser feitas pesquisas e desenvolvimento em questões como descoberta de recursos dos dispositivos pelos agentes e requisitos das aplicações, bem como a estrutura para comportar essas informações.

Com a coleta de informações sobre recursos e requisitos, os mecanismos de negociação e distribuição automática de tarefas são os próximos passos para aumentar a resiliência diante de anomalias ou simplesmente melhorar a QoE. Para isso é necessário alguns mecanismos que servirão de base, por exemplo, mecanismo de descoberta e *heartbeat*. O mecanismo de negociação pode fazer uso de técnicas de aprendizagem de máquina gerando outro ponto interessante de pesquisa.

Pesquisas em algoritmos distribuídos também devem ser realizadas, por exemplo, sincronização entre tarefas em dispositivos distintos e melhores algoritmos para convergência da tabela de fardo.

São necessárias melhorias no que diz respeito ao desenvolvimento, como tornar o *middleware* independente de sistema operacional e linguagem, criar interface para serialização de objetos nômades, permitir desenvolvimento e reorganização de tarefas em tempo de execução e criar ferramenta gráfica para a especificação de dependências das tarefas.

Ainda no que diz respeito ao desenvolvimento e levando em conta o protótipo apresentado devem ser estudadas *engines* de desenvolvimento de jogos (e.g. Unreal Engine, Unity, CryENGINE) e a integração com o *middleware* para torná-lo uma ferramenta viável na prática.

Por último, os trabalhos relacionados a computação na borda da nuvem, bem como esse, carecem de testes experimentais para analisar a viabilidade de aplicações interativas e multimídia distribuídas entre cliente e nuvem, por conta de questões com as apresentadas na Seção 8.1.

10 CONCLUSÃO

É proposto o uso da negociação de recursos com migração e adaptação de conteúdo, envolvendo a nuvem e os dispositivos do cliente para criar aplicações dinâmicas que se adaptam em tempo de execução. Isso visa maximizar a QoE do usuário aproveitando ao máximo recursos disponíveis não só no dispositivo atual, mas em todo o ambiente em que o usuário se encontra, por exemplo, televisão, computadores de mesa, *smartphone*, nuvem, etc.

A negociação entre os envolvidos permite que aja sintonização dos recursos na presença de anomalias ou mesmo no intuito de permitir escalabilidade da aplicação em tempo de execução. A negociação também possui o propósito de trazer a tona um *energy-aware middleware*, que em tempos de dispositivos móveis é uma característica muito benéfica.

A divisão da proposta em MSP, MEC e *middleware* permite que a aplicação desenvolvida seja independente da nuvem. O *middleware* oferece um alto nível de liberdade para o desenvolvedor, essas liberdades podem ser, por exemplo, na escolha da linguagem, do sistema operacional ou do paradigma de programação.

Ao usar a nuvem é possível usufruir de seus benefícios, como acesso ubíquo, compartilhamento assíncrono, escalabilidade/elasticidade e o modelo de pagamento *pay-as-you-go*.

É proposto também que os componentes tenham uma API clara e padronizada seguindo boas práticas da indústria. A API no *middleware* visa fornecer facilidades no desenvolvimento, como distribuição e paralelismo, sincronização, segurança, uso transparente da infraestrutura em nuvem e QaaS, múltiplos usuários, desenvolvimento em tempo de execução, migração de código, etc. O MEC foi pensado para ser modular e com uma API bem definida permite que o administrador possa escolher o que melhor se alinha com a empresa, por exemplo, o *cluster*, o *load balancer* ou o algoritmo de negociação. Por último, o MSP deve servir de base para que serviços sejam fornecidos usando a infraestrutura fornecida pelo CSP.

É sugerido a criação de um MSP, cujo serviço provido é o acesso a uma infraestrutura para uso de desenvolvedores que queiram criar aplicações usando o *middleware* e usuários interessados em usufruir delas.

Nesse trabalho também é sugerido o uso de *utility computing* para o provisionamento

de QoS visando um *QoS-aware middleware*. Muitas aplicações podem se beneficiar com a garantia de QoS, por exemplo, aplicações interativas e multimídia. E com um modelo de cobrança, isso se torna viável para o provedor.

REFERÊNCIAS

ASHAR, K. **The Next Generation Enterprise: Business as a Service in the Cloud**, 2013. Disponível em: <http://www.kunalashar.com/the-next-generation-enterprise-business-as-a-service-in-the-cloud/>.

BABAOGLU, O.; JELASITY, M. Self-* properties through gossiping. **Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences**, The Royal Society, v. 366, n. 1881, p. 3747–3757, 2008. ISSN 1364-503X.

BADGER, L.; GRANCE, T.; PATT-CORNER, R.; VOAS, J. **Cloud Computing Synopsis and Recommendations**, May 2012. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf>.

BALAN, R.; FLINN, J.; SATYANARAYANAN, M.; SINNAMOHIDEEN, S.; YANG, H.-I. The case for cyber foraging. In: **Proceedings of the 10th Workshop on ACM SIGOPS European Workshop**, 2002. (EW 10), p. 87–92. Disponível em: <http://doi.acm.org/10.1145/1133373.1133390>.

BALIGA, J.; AYRE, R.; HINTON, K.; TUCKER, R. Green cloud computing: Balancing energy in processing, storage, and transport. **Proceedings of the IEEE**, v. 99, n. 1, p. 149–167, Jan 2011. ISSN 0018-9219.

BOHEZ, S.; CONINCK, E. D.; VERBELEN, T.; SIMOENS, P.; DHOEDT, B. Enabling component-based mobile cloud computing with the aiolos middleware. In: **Proceedings of the 13th Workshop on Adaptive and Reflective Middleware**, 2014. (ARM '14), p. 2:1–2:6. ISBN 978-1-4503-3232-3. Disponível em: <http://doi.acm.org/10.1145/2677017.2677019>.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog computing and its role in the internet of things. In: **Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing**, 2012. (MCC '12), p. 13–16. ISBN 978-1-4503-1519-7. Disponível em: <http://doi.acm.org/10.1145/2342509.2342513>.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, v. 25, n. 6, p. 599–616, 2009. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X08001957>.

CHESHIRE, S. **It's the latency, stupid**, 1996. Disponível em: <http://rescomp.stanford.edu/ cheshire/rants/Latency.html>.

CORPORATION, N. **Trademark Application for 'CLOUD COMPUTING'**. 1998. Disponível em: <http://tsdr.uspto.gov/caseNumber=75291765caseType=SERIAL$_N$$Osearch Type$ = $statusSearch$>.

GARFINKEL, S.; ABELSON, H. **Architects of the information society: 35 years of the Laboratory for Computer Science at MIT**, 1999. ISBN 978-0-262-07196-3.

GARRISS, S.; CáCERES, R.; BERGER, S.; SAILER, R.; DOORN, L. van; ZHANG, X. Trustworthy and personalized computing on public kiosks. In: **Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services**, 2008. (MobiSys '08), p. 199–210. ISBN 978-1-60558-139-2. Disponível em: <http://doi.acm.org/10.1145/1378600.1378623>.

GRIMM, R.; DAVIS, J.; LEMAR, E.; MACBETH, A.; SWANSON, S.; ANDERSON, T.; BERSHAD, B.; BORRIELLO, G.; GRIBBLE, S.; WETHERALL, D. System support for pervasive applications. **ACM Trans. Comput. Syst.**, ACM, New York, NY, USA, v. 22, n. 4, p. 421–486, Nov 2004. ISSN 0734-2071. Disponível em: <http://doi.acm.org/10.1145/1035582.1035584>.

ISLAM, S.; GRéGOIRE, J.-C. Giving users an edge: A flexible cloud model and its application for multimedia. **Future Generation Computer Systems**, Elsevier, v. 28, n. 6, p. 823–832, 2012. ISSN 0167-739X. Including Special sections SS: Volunteer Computing and Desktop Grids and SS: Mobile Ubiquitous Computing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X12000143>.

JOHNSON, P. **What Does Cloud Computing 2.0 Look Like?**, 2013. Disponível em: <http://www.datacenterknowledge.com/archives/2013/07/31/what-does-cloud-computing-2-0-look-like/>.

KON, F.; COSTA, F.; BLAIR, G.; CAMPBELL, R. H. The case for reflective middleware. **Commun. ACM**, ACM, New York, NY, USA, v. 45, n. 6, p. 33–38, June 2002. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/508448.508470>.

LIMA, G.; GOULART, G.; SERAFIM, V. **Episódio 84 - Segurança em Games**, 2015. Disponível em: <http://www.segurancalegal.com/2015/09/episodio-84-seguranca-em-games.html>.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**, Sept 2011. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.

MORENO, M. F. **Um framework para provisão de QoS em sistemas operacionais**. Tese (Doutorado) — PUC-Rio, 2002.

NSA-STAFF. Securing the cloud with homomorphic encryption. **The Next Wave**, National Security Agency/Central Security Service (NSA/CSS), v. 20, n. 3, p. 1–4, 2014.

NVIDIA. **NVIDIA SHIELD**. 2013. Disponível em: <http://shield.nvidia.com/>.

NVIDIA. **NVIDIA GameStreaming**. 2014. Disponível em: <http://shield.nvidia.com/game-stream>.

NVIDIA. **NVIDIA GRID**. 2015. Disponível em: <http://shield.nvidia.com/grid-game-streaming>.

PETRE, L. Energy-aware middleware. In: **Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the**, 2008. p. 326–334.

REGALADO, A. **Who Coined 'Cloud Computing'?**, 2011. Disponível em: <http://www.technologyreview.com/news/425970/who-coined-cloud-computing/>.

RIMAL, B.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. In: **INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on**, 2009. p. 44–51.

SADJADI, S. M.; MCKINLEY, P. K. A survey of adaptive middleware. **Michigan State University Report MSU-CSE-03-35**, 2003.

SALCHOW, J. K. K. **Balanceamento de carga: Conceitos básicos**, Dec 2014. Disponível em: <http://www.f5networks.com.br/pdf/white-papers/balanceamento-de-carga-conceitos-basicos-wp.pdf>.

SANAEI, Z.; ABOLFAZLI, S.; GANI, A.; BUYYA, R. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. **Communications Surveys Tutorials, IEEE**, v. 16, n. 1, p. 369–392, First 2014. ISSN 1553-877X.

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **Pervasive Computing, IEEE**, v. 8, n. 4, p. 14–23, Oct 2009. ISSN 1536-1268.

SHARIFI, M.; KAFAIE, S.; KASHEFI, O. A survey and taxonomy of cyber foraging of mobile devices. **Communications Surveys Tutorials, IEEE**, v. 14, n. 4, p. 1232–1243, Fourth 2012. ISSN 1553-877X.

SONY. **Remote Play**. 2006. Disponível em: <https://www.playstation.com/en-gb/explore/ps4/features/remote-play/>.

SONY. **PlayStation Now**. 2014. Disponível em: <https://www.playstation.com/en-us/explore/playstationnow/>.

STEAM. **Steam In-Home Streaming**. 2014. Disponível em: <http://store.steampowered.com/streaming/>.

STRIDE. **Stride Decals**. 2015. Disponível em: <http://stride.digital/pt/sobre/>.

SURIE, A.; PERRIG, A.; SATYANARAYANAN, M.; FARBER, D. Rapid trust establishment for pervasive personal computing. **Pervasive Computing, IEEE**, v. 6, n. 4, p. 24–30, Oct 2007. ISSN 1536-1268.

TAURION, C. **Estimando o valor da Computação em Nuvem**, 2010. Disponível em: <http://imasters.com.br/artigo/19195/cloud/estimando$_{ov}alor_{d}a_{c}omputacao_{e}m_{n}uvem$/>.

ZHANG, J.; FIGUEIREDO, R. Application classification through monitoring and learning of resource consumption patterns. In: **Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International**, 2006. p. 10.

ZHU, W.; LUO, C.; WANG, J.; LI, S. Multimedia cloud computing. **Signal Processing Magazine, IEEE**, v. 28, n. 3, p. 59–69, May 2011. ISSN 1053-5888.