Pós-Graduação em Ciência da Computação

Iúre de Sousa Fé

PLANEJAMENTO DE TRANSCODIFICAÇÃO DE VÍDEO EM NUVEM ELÁSTICA

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Iúre de Sousa Fé

PLANEJAMENTO DE TRANSCODIFICAÇÃO DE VÍDEO EM NUVEM ELÁSTICA

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Paulo Romero Martins Maciel*

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

F288p Fé, Iúre de Sousa
Planejamento de transcodificação de vídeo em nuvem elástica / Iúre de Sousa Fé. – 2017.
107 f.: il., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndices.

1. Ciência da computação. 2. Computação em nuvem. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004 CDD (23. ed.) UFPE- MEI 2017-109

Iúre de Sousa Fé

Planejamento de Transcodificação de Vídeo em Nuvem Elástica

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

Aprovado em: 03/02/2017

BANCA EXAMINADORA

Prof. Dr. José Augusto Suruagy Monteiro
Centro de Informática / UFPE

Prof. Dra. Patricia Takako Endo
Universidade de Pernambuco - Campus Caruaru

Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática / UFPE
(**Orientador**)

Eu dedico esta dissertação a Deus, minha família, amigos e professores que me deram o apoio necessário para alcançar meus objetivos.

Agradecimentos

Agradeço a toda minha família, em especial a minha irmã, Olga, por seu apoio em toda a minha vida e por trazer mais um integrante a família, o Heitor. Também agradeço à minha namorada, Driele, pela paciência, felicidade, cumplicidade e ajuda em toda trajetória deste trabalho. Meus mais sinceros agradecimentos ao professor Dr. Paulo Maciel pela orientação, conhecimentos transmitidos e exemplo de dedicação ao trabalho.

Meu obrigado a todos os amigos do grupo MoDCS, em especial ao Jamilson e Rubens pela colaboração neste trabalho. Também agradeço à amizade de todos da minha turma, Aline, Camila, Erico. Meu obrigado ao Eric e Jonas, pelas horas de conversas e caminhadas, que tornaram minha estadia em Recife bem mais saudáveis. Agradeço ao meu amigo de longa data, Márcio, pelos momentos de descontração nesta cidade. Meu agradecimento a Lucília e Leônidas, pelo apoio e amizade. Obrigado aos meus chefes e colegas de trabalho por proporcionarem a oportunidade de realizar este trabalho.

Por fim, agradeço a Deus por me dar forças e saúde para completar este trabalho.

The number one benefit of information technology is that it empowers people to do what they want to do. It lets people be creative. It lets people be productive. It lets people learn things they didn't think they could learn before, and so in a sense it is all about potential.

—STEVE BALLMER

Resumo

O tráfego de vídeo ocupa a maior parte do volume de dados que são transmitidos pela Internet, principalmente devido à capacidade de usuários comuns gravarem e compartilharem seus próprios conteúdos. Entretanto, há uma grande heterogeneidade de dispositivos, softwares e rede para visualização dessas mídias, requerendo que esses vídeos sejam transcodificados para formatos compatíveis com a maioria dos visualizadores de vídeo. Transcodificar vídeos é uma atividade computacionalmente cara e com demanda altamente variável, portanto, pode beneficiar-se da capacidade distribuída e elástica da computação em nuvem. No entanto, identificar uma configuração dos mecanismos automáticos de elasticidade (*auto-scaling*), que cumpra os requisitos mínimos de desempenho requeridos pelo SLA ao menor custo possível não é uma tarefa simples, requer ajustar diversos parâmetros, como o momento de criar e retirar VMs, o tipo de VM que será adicionada, levando em conta o tempo de transcodificação e instanciação de cada tipo. Além disso, em infraestruturas públicas, os contratos firmados com o provedor de nuvem podem apresentar custos diferentes para um mesmo tipo de VM alugada, onde a opção apropriada é relacionada com o *auto-scaling* e carga de trabalho esperada. Já em infraestruturas de nuvem privadas, a complexidade de configuração trará também aspectos do dimensionamento dinâmico adequado da infraestrutura física para reduzir o consumo elétrico enquanto cumpre o SLA. Com objetivo de auxiliar na escolha desses parâmetros, esta dissertação propõe modelos em Redes de Petri Estocástica para computar a vazão, o tempo médio de resposta, o custo em nuvens públicas e o consumo elétrico em nuvens privadas. Essas métricas são avaliadas a partir da entrada dos parâmetros de configuração e taxa de requisições esperada para o sistema. Os modelos estocásticos propostos também foram integrados com o algoritmo de otimização GRASP, com objetivo de encontrar as configurações que devem ser adotadas na nuvem para cumprir o SLA e minimizar o custo de manter o sistema. Os estudos de caso demonstram que a combinação dos modelos com mecanismos de otimização é útil para orientar os administradores nas escolhas dos valores dos parâmetros de configuração para implantar e ajustar sistemas, respeitando os requisitos de desempenho e minimizando o custo. A aplicação dessa abordagem também permitiu identificar o comportamento do custo de um sistema em relação ao SLA, em um dos estudos de caso apresentados, a redução de 30 para 15 segundos no tempo de resposta mínimo representou um aumento de 299% no custo, já uma redução de 45 para 30 segundos apenas um aumento de 6%. Este comportamento é especialmente útil na negociação de novos SLAs.

Palavras-chave: Transcodificação de Vídeo. Computação em Nuvem. Auto-Scaling. Desempenho. Modelagem Estocástica. Otimização

Abstract

Video traffic occupies most of the volume of transmitted data over the Internet, mainly because of the ability of users to record and share their content. However, there is a high heterogeneity of devices, software, and network for viewing these media, requiring these videos to be transcoded to formats compatible with most video viewers. Transcoding videos is a computationally expensive and highly variable demand activity so that it can benefit from the distributed and elastic ability of cloud computing. Although, identifying a configuration of the automatic elastic mechanisms (*auto-scaling*), which meets the minimum performance requirements required by the SLA at the lowest possible cost is not a simple task. It needs adjusting several parameters, such as the time to create and remove VMs, the type of VM that will be added, taking into account the transcoding and instantiation time of each type. Besides, in public infrastructures, contracts with the cloud provider may incur different costs for the same type of hired VM, where the appropriate option is related to *auto-scaling* and expected workload. In private cloud infrastructures, configuration complexity will also bring aspects of the proper dynamic dimensioning of the physical infrastructure to reduce electrical consumption while complying with the SLA. To assist in the choice of these parameters, this dissertation proposes Stochastic Petri Nets models to compute the throughput, the mean response time, and the cost in public clouds and electrical consumption in private clouds. These metrics are evaluated from the input of configuration parameters and requisition rate expected for the system. The proposed stochastic models were also integrated with the GRASP optimization algorithm, in order to find the configurations that should be adopted in the cloud to comply with the SLA and minimize the cost of owning the system. Case studies demonstrate that the combination of models with optimization mechanisms is useful to guide administrators in choosing the values of configuration parameters to deploy and tune systems while respecting performance requirements and minimizing cost. The application of this approach also allowed to identify the behavior of the cost of a system in relation to the SLA, in one of the presented case studies, the reduction of 30 to 15 seconds in the minimum response time represented a 299% increase in cost, while a reduction of 45 to 30 seconds only an increase of 6%. This behavior is especially useful when negotiating new SLAs.

Keywords: Video Transcoding. Cloud Computing. Auto-Scaling. Performance. Stochastic Modeling. Optimization

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

SSH Secure Shell 78

UML Unified Modeling Language 42

VBR Variable Bit Rate 37

VLE Virtual Learning Environment 38

VM Virtual Machine 19

VND Variable Neighborhood Descent 36

VNS Variable Neighborhood Search 36

VoD Video On Demand 37

Wol Wake-on-LAN 67

Sumário

1

Introdução

O tráfego na Internet cresceu muito nos últimos anos, em 2020 é esperado que seja 3 vezes maior que em 2015 (CISCO SYSTEMS, 2016). Nessa estimativa, o tráfego de vídeo ocupa a maior parcela, expandindo de 67% em 2015 para 80% do tráfego total em 2020 (CISCO SYSTEMS, 2016). Um dos responsáveis é o compartilhamento de vídeos entre os usuários, que tem se tornado cada vez mais comum em plataformas como Facebook, Twitter, Youku (o site mais popular de compartilhamento de vídeos na China) e Youtube, esse último, é visitado por mais de um terço dos usuários de Internet, sendo mais da metade, através de dispositivos móveis, além de possuir mais usuários de 18 a 49 anos que qualquer TV a cabo nos EUA (YOUTUBE, 2016).

No entanto, requisitos de compatibilidade devem ser atendidos, pois os usuários compartilham vídeos de diferentes conteiner e codecs, mas nem todos os formatos são suportados pelos diversos dispositivos e browsers. Para melhorar a compatibilidade, algumas recomendações surgiram, como o HTML5, onde foi recomendada a utilização dos formatos MP4, WebM, ou Ogg para que os vídeos sejam suportados pelos *browsers* mais usados (W3C, 2010; W3SCHOOLS, 2016). Além disso, técnicas como *Adaptive Bit Rate - ABR*, requerem que um mesmo conteúdo seja codificado em vários formatos, para que possa ser exibido nos variados dispositivos e condições de rede.

Diante disso, surge a necessidade da transcodificação dos vídeos enviados pelos usuários para os containers e codecs suportados nas mais diversas plataformas. Porém, a transcodificação é uma atividade computacionalmente cara, demandando grande quantidade de recursos, principalmente em sistemas na Internet, onde diversas requisições simultâneas podem ser realizadas (YANG et al., 2015). O volume dessas requisições também costuma ser variável, com momentos de alta e baixa demanda. Nesse contexto, para manter um nível adequado de serviço, seria necessário adquirir uma infraestrutura grande o suficiente para atender a demanda no pico de requisições, por outro lado, haverá o desperdício pelo superprovisionamento quando a demanda for baixa.

Muitos autores utilizam computação em nuvem para solucionar esse problema (LAO; ZHANG; GUO, 2012; APARICIO-PARDO et al., 2015; TIMMERER et al., 2016), pois a

computação em nuvem é um paradigma capaz de prover recursos computacionais sob demanda através da elasticidade. Esses recursos devem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento (MELL; GRANCE, 2011).

A Figura 1.1 evidencia a diferença entre uma infraestrutura fixa e uma elástica na nuvem. O gráfico da esquerda apresenta uma infraestrutura fixa, onde nas regiões vermelhas foram adquiridas mais capacidade que o necessário, gerando desperdício de recurso. Já na região verde, a capacidade existente não foi capaz de atender a demanda. O uso de mecanismos elásticos é apresentado no gráfico da direita, nele podemos ver a capacidade sempre levemente maior que a demanda, portanto atendendo a necessidade de recursos e evitando custos desnecessários.

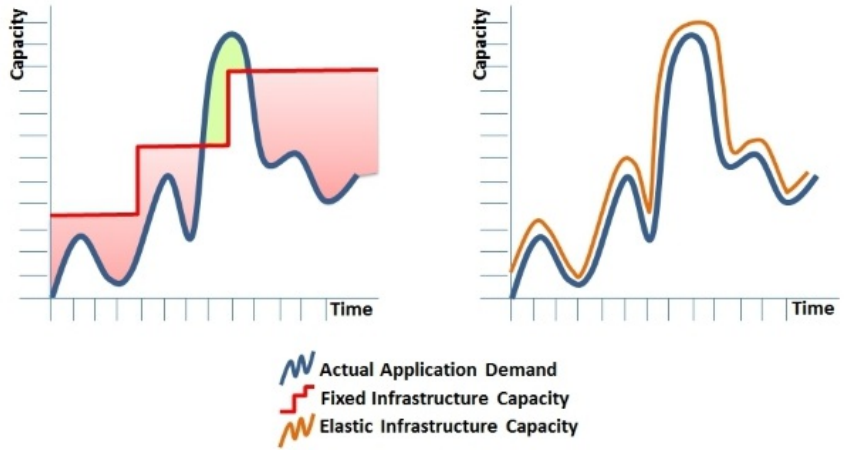


Figura 1.1: Comparação de insfraestruturas fixas e elásticas (WADIA, 2016)

As fontes dos custos para a implantação de sistemas na nuvem irão depender dos modelos de implantação utilizados. Esses modelos usualmente são classificados em públicos ou privados. Nos públicos, a infraestrutura é provida por uma organização para muitos clientes, nessas nuvens os custos variam em relação aos contratos e tempo de utilização dos recursos. Já as nuvens privadas são implantadas para uso exclusivo de uma única organização, os custos são relacionados à aquisição de infraestrutura, operações de manutenção, consumo elétrico, e etc.

A Figura 1.2 retrata os envolvidos no sistema de transcodificação de vídeo na nuvem, onde os usuários enviam vídeos nos mais diferentes formatos que devem ser codificados pelo conjunto de VMs contratadas para os formatos acessíveis pelas diversas plataformas. O administrador do sistema deve considerar a variação da demanda e selecionar os parâmetros e contratos da nuvem com objetivo de cumprir o acordo de nível de serviço *(Service level agrement - SLA)* ao menor custo possível. Entretanto, existem inúmeros valores possíveis para cada parâmetro, e a escolha de um terá influência na escolha dos outros, portanto, selecionar adequadamente os parâmetros de configuração da nuvem não é uma tarefa simples, e má configuração pode levar ao não cumprimento do SLA ou custos desnecessários.

A transcodificação de vídeo na nuvem foi discutida por diversos trabalhos. Os autores LAO; ZHANG; GUO (2012) propuseram métodos de paralelização da transcodificação na nuvem para reduzir o tempo de resposta. Modelos são uma ferramenta de auxílio no planejamento de

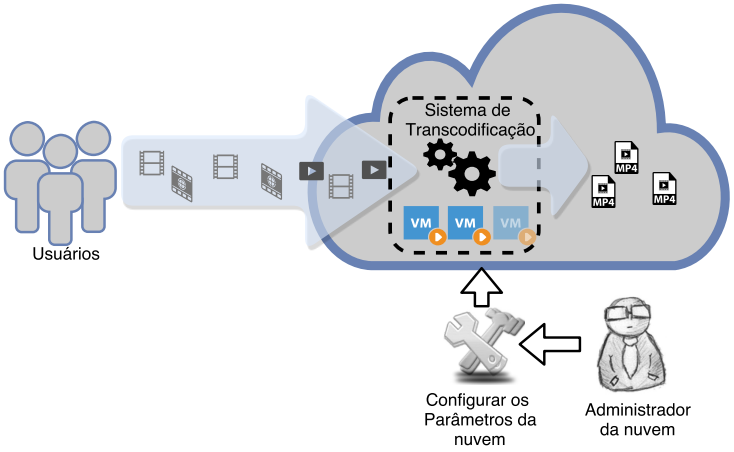


Figura 1.2: Sistema de transcodificação de vídeo

elementos do *auto-scaling*. Os autores RIBAS et al. (2015) compararam através de modelos diversos contratos em nuvens públicas.

Esta dissertação propõe auxiliar no planejamento de sistemas de transcodificação de vídeos em nuvens através de modelos estocásticos. Será considerado o uso de nuvens públicas e privadas. Em nuvens públicas o custo será dado: pela contratação de VMs reservadas; tempo de utilização de instâncias sob demanda para a taxa de chegada esperada; e a configuração da elasticidade definida pelo usuário. Em nuvens privadas o custo levará em conta o consumo elétrico de manter a infraestrutura de computação ligada; o consumo de cada *Virtual Machine - VM* para realizar transcodificações; e os parâmetros de elasticidade. Os modelos gerados também foram utilizados como entrada para um algoritmo de otimização que buscou configurações que cumprissem o SLA ao mesmo tempo que minimizassem o custo.

1.1 Motivação e Justificativa

O efetivo compartilhamento de vídeos num cenário de alta heterogeneidade de softwares, dispositivos e redes, requer a transcodificação dessas mídias para formatos que se adequem a cada usuário. Uma vez que a transcodificação é uma atividade computacionalmente custosa, ela tem migrado para nuvem com objetivo de reduzir custos por meio de mecanismos elásticos da nuvem para se adaptar a uma demanda com períodos e picos e baixa procura. Empresas como a Encoding.com, utilizam a infraestrutura da nuvem pública da Amazon para alto volume de transcodificação de mídias digitais, chegando a 40,000 arquivos por dia (AMAZON, 2016; ENCODING.COM, 2016).

Essa abordagem dá a possibilidade de pagar apenas os recursos consumidos (*pay-per-use*). O usuário da nuvem pode adicionar e retirar recursos unilateralmente através de procedimentos manuais ou automáticos. O *auto-scaling* é o método de ajuste automático da quantidade de recursos, tornando possível atingir o desempenho esperado pela aplicação hospedada na nuvem enquanto a carga de trabalho sofre variação.

Normalmente o mecanismo de *auto-scaling* monitora a aplicação e compara com limiares para a instanciação e destruição de máquinas virtuais (VMs). Entretanto, a definição desses limiares é uma tarefa bastante complexa (BISWAS et al., 2015), pois sua definição se relaciona com a configuração de diversos outros parâmetros, contratos e a carga de trabalho. Então, a principal motivação desse trabalho é devida a essa dificuldade de identificar quais valores devem ser utilizados nos parâmetros de configuração do sistema de transcodificação na nuvem com *auto-scaling* para que seja possível cumprir o SLA ao menor custo possível.

Dentre as características dos parâmetros que foram considerados nesse trabalho, os limiares devem levar em conta que a VM não se torna operacional no momento que é requisitada, pois levará tempo para criar a VM e iniciar o sistema operacional e a aplicação. Também deve ser levado em conta o tipo de VM que será instanciada no *auto-scaling*, cada tipo, possui capacidade de processamento, quantidade de memória e custos diferentes. Uma seleção adequada permite transcodificações mais rápidas, ou até mesmo a possibilidade de suportar mais trabalhos simultâneos. Além disso, existem diferenças entre contratos para um mesmo tipo de VM, alugar VMs por longos períodos de tempo (conhecido como VMs reservadas) possuem custos diferentes do aluguel por demanda (VMs elásticas ou sob demanda), que é pago por curtos períodos de tempo (normalmente por hora de uso). O uso de VMs elásticas não é vantajoso quando são usadas por longos períodos de tempo, mas é muito útil para suprir demandas transitórias. A utilização adequada das VMs elásticas é alcançado pelo ajuste configuração dos limiares de instanciação e destruição para cada tipo de VM.

No caso de nuvens privadas, não há contratos de VMs reservadas e elásticas, porém manter VMs ligadas sem uso irá aumentar o consumo elétrico, tanto pelo consumo de manter uma VM ociosa, quanto o de manter hardwares de computação ligados para sustentar essas VMs. Já a escolha do tipo de VM irá definir o consumo e o desempenho máximo daquela VM e também a quantidade total de VMs que a infraestrutura existente pode suportar.

Há também outros parâmetros como: quantidade de transcodificações por VM, quantidade de VMs a serem requisitadas a cada limiar de instanciação; limiares para ligar e desligar nós de computação; quantidade de nós de computação ligados. A definição inadequada dos parâmetros de configuração do *auto-scaling* pode levar à violação dos SLAs de desempenho e/ou desperdício de recursos.

1.2 Objetivos

A transcodificação de vídeo é uma atividade computacionalmente custosa e necessária para o compartilhamento de vídeos na Internet. Computação em nuvem, quando adequadamente ajustada, oferece uma capacidade elástica capaz de atender a demanda variável de usuários, reduzindo os custos e cumprindo os requisitos de desempenho. Porém, identificar uma configuração que atenda aos requisitos de desempenho e custo é um desafio.

A proposta dessa dissertação é auxiliar no planejamento de sistemas de transcodificação

de vídeo em nuvens públicas e privadas por meio de modelos estocásticos e otimização. Os modelos devem receber como entrada os diversos parâmetros de configuração, contratos e carga de trabalho e apresentem como resultado, as métricas de desempenho e custo. A otimização auxiliará no planejamento ao receber os modelos gerados as restrições impostas pelo SLA e oferecerá ao administrador de sistemas o valor de cada parâmetro para que o *auto-scaling* ofereça apenas os recursos necessários para cumprir o nível de serviço, portanto, minimizando o custo.

Para alcançar esse objetivo é necessário cumprir os seguintes objetivos específicos:

- Criar e validar um modelo de desempenho e custo de utilização de VMs de nuvem pública envolvendo *auto-scaling*;
- Criar e validar um modelo de desempenho e consumo elétrico de uma nuvem privada com *auto-scaling* envolvendo o consumo elétrico das VMs e servidores;
- Desenvolver algoritmos de otimização contemplando os dois modelos de nuvem;
- Aplicar os modelos e a otimização em estudos de caso visando identificar o comportamento das métricas em relação aos parâmetros e SLAs;

1.3 Estrutura da Dissertação

Este documento está dividido em seis capítulos. Após este capítulo, o Capítulo 2 apresenta os conceitos básicos no qual este trabalho está envolvido e também os trabalhos relacionados. O Capítulo 3 apresenta em detalhes a metodologia de avaliação proposta e as arquiteturas relacionadas. O Capítulo 4 insere os modelos, dando ênfase aos parâmetros envolvidos no processo, também apresenta a validação desses modelos pela comparação das suas métricas com as obtidas por sistemas reais. O Capítulo 5 apresenta estudos de caso baseados nos modelos propostos. Por fim, no Capítulo 6 apresentamos as considerações finais, listando as contribuições, e apresentando direcionamentos futuros.

2

Fundamentação teórica

Este capítulo apresenta os conceitos fundamentais para o entendimento desse trabalho. Será inicialmente apresentado sobre *Auto-scaling* em nuvem, bem como mecanismos de utilização dessa técnica, para então, uma introdução sobre modelagem de desempenho através de Stochastic Petri Net - SPNs. Em seguida, abordaremos estratégias de modelagem de potência elétrica. Também será apresentada a meta-heurística de otimização Greedy Randomized Adaptive Search Procedure - GRASP. Por fim, apresentaremos uma introdução sobre transcodificação de vídeo.

2.1 Auto-scaling

O NIST define que uma das principais características da computação em nuvem é a capacidade do cliente unilateralmente alocar recursos computacionais sem interação humana com o provedor de serviço (*on-demand self-service*). Outra importante característica é a rápida elasticidade (*rapid elasticity*), que é a capacidade de adicionar e retirar recursos (automaticamente ou não) para se adaptar à demanda (MELL; GRANCE, 2011). Essas características dão origem às técnicas de *auto-scaling*, que proveem uma solução automática para alocação de recursos. O *auto-scaling* vem sendo estudado em diversos trabalhos (ASSUNÇÃO et al., 2016; XIAO; CHEN; LUO, 2014; BISWAS et al., 2015; LORIDO-BOTRÁN; MIGUEL-ALONSO; LOZANO, 2012).

Configurar o *auto-scaling* normalmente depende do mapeamento dos recursos necessários para cumprir as demandas de desempenho requeridas por cada aplicação específica. Portanto, é necessário identificar a quantidade de cada recurso que deve ser adquirida e liberadas de acordo com a variação da demanda (LORIDO-BOTRÁN; MIGUEL-ALONSO; LOZANO, 2012). Os autores em (GHANBARI et al., 2011) propõem uma lista de subcomponentes da infraestrutura para identificação das necessidades de recursos da aplicação:

- *Hardware*: utilização da CPU, uso de disco, uso de memória, uso da interface de rede;
- Processos de sistema operacional: tempo de CPU, falta de página;

- Balanceador de carga: tamanho da fila, taxa de entrada de sessões, número de sessões, número de sessões bloqueadas, número de erros;
- Servidor Web: número de requisições, número de requisições em estados específicos (fechada, enviando, em espera, etc.);
- Servidor de Aplicação: quantidade de *threads* totais, uso de memória, quantidade de sessões, requisições processadas, requisições pendentes, tempo de resposta;
- Banco de dados: número de *threads* ativas, número de transições em estados específicos (escritas, *commit*, *roll-back*, ...).

Além de identificar a origem da necessidade de recursos, a configuração do *auto-scaling* também requer identificar o instante para a aquisição e destruição desses recursos. As técnicas para identificar esse instante se dividem principalmente em preditivas e reativas. As técnicas preditivas tentam antecipar as necessidades futuras de recurso da aplicação, e então, adquirir ou liberar VMs antecipadamente. Normalmente essa técnica utiliza séries temporais, analisando a necessidade de recursos anteriores para encontrar padrões repetíveis e então extrapolar futuras demandas. Isso visa garantir que os recursos necessários sempre estejam disponíveis ao mesmo tempo em que tenta evitar manter recursos quando não necessário. Essa técnica também pode ser utilizada em combinação com a reativa (IQBAL et al., 2011).

Por outro lado, as técnicas reativas são baseadas em regras e respondem a mudanças no sistema quando eles encontram limiares predefinidos (LORIDO-BOTRÁN; MIGUEL-ALONSO; LOZANO, 2012). Sendo esta a técnica mais amplamente utilizada em sistemas comerciais (GALANTE; BONA, 2012). No caso da Amazon, o provedor de aplicações pode monitorar recursos e logs para encontrar limiares e adicionar e retirar VMs. Além disso, a API (Amazon Elastic Compute Cloud - EC2) pode ser utilizada para gerenciar remotamente a infraestrutura virtual, recuperando informações das instâncias, deletando e criando novas VMs, entre outras atividades, sem que haja a necessidade de acessar diretamente o painel de gerenciamento da nuvem, oferecendo capacidade para o desenvolvimento de mecanismos de *auto-scaling* customizados.

O *auto-scaling* reativo requer a definição de pelo menos dois limiares, o de instanciação e o de destruição. O limiar de instanciação é responsável pelo *scaling-up*, que é a adição de recursos no sistema. Já o limiar de destruição é responsável pelo *scaling-down* que retira recursos da infraestrutura, reduzindo os custos operacionais da aplicação. A definição desses limiares deve levar em conta que a adição de recursos não é instantânea. Uma nova VM requer além do tempo para ser criada, o tempo para que sejam iniciados o sistema operacional e a pilha de software necessários para aplicação. Portanto, deve-se adicionar recursos antes que o desempenho da aplicação atinja seu limite mínimo aceitável.

É importante destacar que as nuvens públicas costumam oferecer pelo menos dois tipos de contrato para máquinas virtuais, as reservadas e as sob demanda. As VMs reservadas são

contratadas por longos períodos de tempo fixos, normalmente superiores a um ano. Já as VMs adicionadas pelo mecanismo de *auto-scaling* são VMs sob demanda (*on-demand*), o seu preço é relacionado ao tempo de uso. Entretanto, as VMs sob-demanda são mais caras se utilizadas por longos períodos de tempo, sendo úteis apenas para responder a picos transitórios da carga de trabalho.

Outro fator a ser configurado no *auto-scaling* é o tipo de VM que será criada. Em nuvens públicas, VMs com maiores quantidades de recursos (número de núcleos de vCPUs, e quantidade de memória) têm maior custo. Porém, esse custo pode ser justificado pelo menor tempo de processamento e a capacidade de suportar maior número de atividades concorrentes.

Em resumo, a configuração adequada do *auto-scaling* irá depender de vários fatores, como: dos limiares de destruição e instanciação de VMs; *step-size*; tipos de VM; e quantidades de VM por contrato. Esses parâmetros devem ser escolhidos levando em conta que eles se inter-relacionam, tornando essa tarefa bastante difícil e a escolha errada pode levar ao não cumprimento do SLA ou à custos desnecessários.

2.2 Modelagem de Desempenho por SPN

Avaliação de desempenho pode ser realizada através de medidas no sistema real ou por meio de modelos que representam as características e comportamento do sistema. Em muitos casos, modelagem é o método escolhido, seja por que o sistema não está completamente construído, ou devido às dificuldades intrínsecas de criar cenários e analisar o sistema real. Modelos são uma poderosa ferramenta para testar novos conceitos, novas políticas de operação, ou verificar a utilização de recursos para então implementar melhorias no sistema real (CHUNG, 2003).

A seguir serão apresentados as técnicas necessárias para a computação da análise numérica em SPN, inicialmente serão apresentados de CTMCs e como é realizado o seu cálculo do vetor de probabilidade estacionária e posteriormente a descrição formal e gráfica das SPNs.

2.2.1 CTMC

Cadeias de Markov são amplamente utilizados para modelar as características de desempenho e dependabilidade de sistemas. Elas levam em conta a interação dos vários componentes de uma arquitetura. Um modelo de Markov pode ser descrito como um diagrama de espaço de estados associado a um processo de Markov, que é uma subclasse de um processo estocástico (BOLCH et al., 2006).

Um processo estocástico é um conjunto de variáveis aleatórias $\{X_t : t \in T\}$ onde cada variável aleatória X_t é indexada pelo parâmetro $t \in T$, o qual é usualmente chamado de parâmetro temporizado se $T \subset R_+ = [0, \infty)$, ou seja, T é um conjunto de números reais não negativos. O conjunto de todos os possíveis valores de X_t (para cada $t \in T$) é chamado como espaço de estados

S do processo estocástico (BOLCH et al., 2006).

Sendo $Pr\{k\}$ a probabilidade de um dado evento k ocorrer. Um processo de Markov é um processo estocástico no qual $Pr\{X_{t_{n+1}}\}$ depende apenas do valor anterior de X_{t_n}, para todos $t_0 > t_1 > ...t_n > t_{n+1} >$, e todo $s_i \in S$. O que significa que a evolução de um processo de Markov é descrito apenas pelo estado atual, e é independente dos estados passados, sendo referenciado também como processo sem memória (HAVERKORT, 2001).

As principais abstrações aplicadas no processo de formalização são as associações das durações das atividades com variáveis aleatórias e as inclusões de probabilidades para representar alternativas na evolução dos sistemas. Dependendo do tipo de variáveis aleatórias usadas para representar as durações das atividades no sistema podem ser utilizadas as interpretações de cadeias de Markov de tempo discreto (*Discrete Time Markov Chain - DTMC*) ou de tempo contínuo (*Continuous Time Markov Chain - CTMC*) (BOLCH et al., 2006). Devido à propriedade de Markov, o tempo entre as atividades deve seguir uma distribuição sem memória, no caso da CTMC é usada a distribuição exponencial.

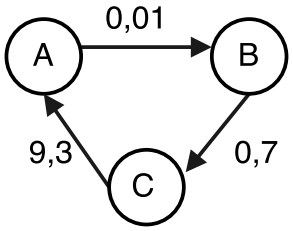


Figura 2.1: Exemplo de CTMC

As CTMCs podem ser representadas graficamente por um grafo direcionado com transições nomeadas, indicando a taxa que a transição ocorre de um estado para outro. Um exemplo de uma CTMC pode ser vista na Figura 2.1. A CTMC também pode ser presentada pela sua matriz de transição, ou matriz geradora, onde os elementos representam as taxas de mudança entre os estados da cadeia. A matriz geradora Q é composta por componentes q_{ii} e q_{ji}, onde $i \neq j$, e $\sum q_{ij} = -q_{ii}$. Considerando a CTMC da Figura 2.1 e o espaço de estados $S = \{A,B,C\} = \{0,1,2\}$, a matriz Q pode ser vista abaixo:

$$Q = \begin{pmatrix} q_{00} & q_{01} & q_{02} \\ q_{10} & q_{11} & q_{12} \\ q_{20} & q_{21} & q_{22} \end{pmatrix} = \begin{pmatrix} -0,01 & 0,01 & 0 \\ 0 & -0,7 & 0,7 \\ 9,3 & 0 & -9,3 \end{pmatrix}$$

As Equações 2.1 permitem a computação dos vetor de probabilidade de estado para análise de estado estacionário. Possuindo o vetor de probabilidade de estado as métricas de desempenho podem ser derivadas para cada sistema que é modelado.

$$\pi \times Q = 0, \sum_{i \in S} \pi_i = 1 \tag{2.1}$$

Explicações detalhadas de como gerar a CTMC de uma SPN podem ser obtidas em (HAVERKORT, 2001; BOLCH et al., 2006). Os valores das métricas por meio da análise

numérica neste trabalho, são obtidas através do espaço de estados CTMC obtido da SPN e posterior resolução do sistema de Equações 2.1.

2.2.2 Redes de Petri

Redes de Petri (*Petri Net - PN*) são uma família de formalismos baseados em estado, apropriada para modelar diversos tipos de sistemas que possuam mecanismos de concorrência, assincronicidade, distribuição, determinísticos, ou estocásticos. Sua representação gráfica permite a visualização de atividades concorrentes e dinâmicas do sistema. Enquanto suas características matemáticas tornam possíveis criar equações de estado, equações algébricas ou outros modelos matemáticos que regem o comportamento do sistema (MURATA, 1989). Desde sua proposição em (PETRI, 1962), muitas extensões foram propostas para permitir representar características de sistemas não representadas nos primeiros modelos. Essas extensões adicionaram elementos temporizados, orientados a objetos, coloridas, entre outras.

A representação gráfica das PNs são formadas por: lugares (Figura 2.2 (a)) que correspondem às variáveis de estado; transições (Figura 2.2 (b)) que representam as ações e eventos do sistema; arcos (Figura 2.2 (c)) que presentam os fluxos de marcas pelo sistema; e marcas (*tokens*) (2.2 (d)), cujo conjunto representa o estado do sistema em um instante. A realização de uma ação ou evento (disparo de uma transição) no sistema está ligado a pré-condições, deve existir uma relação entre os lugares e a transição para que esta possa ou não realizar a ação (disparar). Após o disparo alguns lugares terão suas informações alteradas (Marcas), ou seja, levará a uma pós-condição.

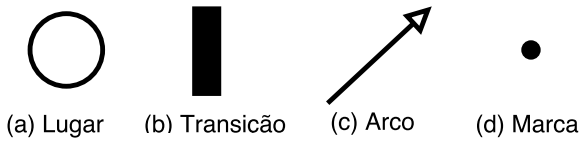


Figura 2.2: Elementos de uma Rede de Petri

A Figura 2.3 contém a representação gráfica de um modelo em três estados diferentes. Ele possui três lugares (**P1**, **P2** e **P3**) e duas transições (**T1** e **T2**). No estado inicial (Figura 2.3 (a)), P1 possui quatro *tokens* e as precondições para o disparo de **T1** estão satisfeitas, isto é, há *tokens* em **P1** e **P2**. Com o disparo de **T1** o modelo vai para o segundo estado (Figura 2.3 (b)), nesse estado, **T1** não está mais habilitada para o disparo, porém **T2** está. O disparo de **T2** leva o modelo ao estado (Figura 2.3 (c)) que habilita novamente **T1**.

A introdução da noção de tempo em PNs foi introduzida nos trabalhos de MERLIN; FARBER (1976) e NOE; NUTT (1973). O tempo para realização dos eventos temporizados pode ser intervalar, determinístico, não determinístico ou estocástico (RAMCHANDANI, 1974; MERLIN; FARBER, 1976; AJMONE MARSAN; CONTE; BALBO, 1984).

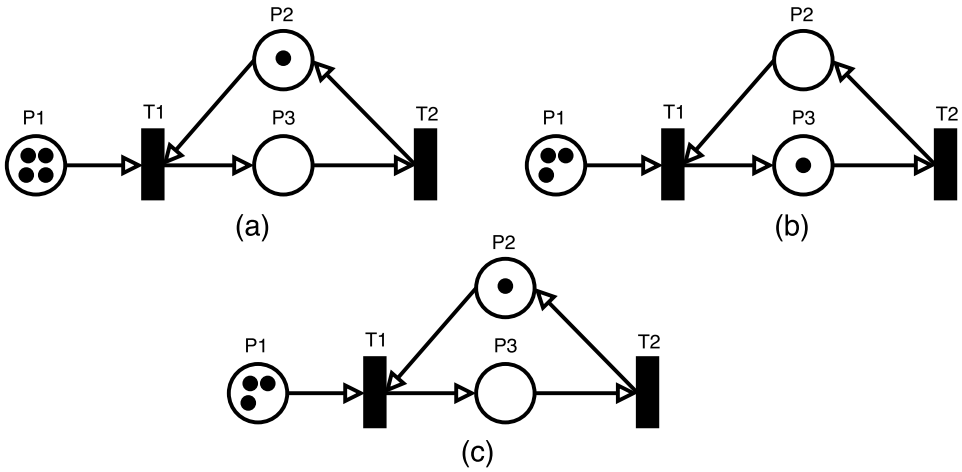


Figura 2.3: Exemplo de uma Rede de Petri

2.2.3 Rede de Petri Estocástica

A introdução de tempos estocásticos resultou na Rede de Petri Estocástica SPN, que permitiu ligar as PNs a avaliação de desempenho, uma área normalmente baseada na abordagem de modelagem estocástica (MOLLOY, 1981). Outra extensão foi a adição de tempos estocásticos combinados com tempos nulos que permitiram modelar sistemas com tempos e condições lógicas. O paradigma resultante foi nomeado de SPNs generalizadas (Generalized Stochastic Petri Net - GSPN) (AJMONE MARSAN; CONTE; BALBO, 1984), por concisão o acrônimo SPN é frequentemente usado para representar toda a família de modelos derivados da SPN (GERMAN, 2000), como será utilizado neste trabalho.

Nas SPNs, as atividades que têm tempos associados, são representadas por transições temporizadas, que são representadas por retângulos brancos (Figura 2.4 (a)). Nas SPNs o período de habilitação da transição corresponde ao tempo para realizar a atividade, e o disparo ao fim da atividade. As transições imediatas não têm tempo associado, e possuem prioridade de disparo maior que as transições temporizadas. Podendo também possuir prioridades e probabilidades entre transições imediatas diferentes (BRINKSMA; HERMANNS; KATOEN, 2003; MARSAN et al., 1998).

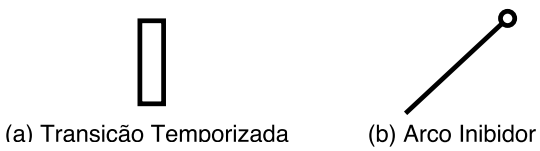


Figura 2.4: Elementos adicionais em uma SPN

Outra adição posterior das PNs, são os arcos inibidores (Figura 2.4 (b)), essa adição permite testar se um lugar não possui *tokens*. Com a presença do arco a transição estará habilitada se a quantidade de *tokens* no lugar p associado ao arco for menor que o peso do arco n, ou seja $M(p) < n$ (MARSAN et al., 1994).

Adotaremos a definição formal de SPNs segundo (GERMAN, 2000), que é apresentada a seguir: Uma SPN é definida pela 9-tupla SPN = (P,T, I,O,H,Π,G,M_0,Atts), onde:

- $P = \{p_1, p_2, ..., p_n\}$ é o conjunto de lugares. n é a quantidade de lugares;
- $T = \{t_1, t_2, ..., t_m\}$ é o conjunto de transições imediatas e temporizadas, $P \cap T = \emptyset$. m é a quantidade de transições;
- $I \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de entrada (que podem ser dependentes de marcações);
- $O \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de saída (que podem ser dependentes de marcações);
- $H \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ é a matriz que representa os arcos de inibidores (que podem ser dependentes de marcações);
- $\Pi \in \mathbb{N}^n$ é o vetor que associa o nível de prioridade a cada transição;
- $G \in (\mathbb{N}^n \to \{true, false\})^n$ é o vetor que associa uma condição de guarda relacionada à marcação do lugar a cada transição;
- $M_0 \in (\mathbb{N}^n)$ é o vetor que associa uma marcação inicial de cada lugar (estado inicial);
- $Atts = (Dist, Policy, Concurrency, W)^m$ compreende o conjunto de atributos associados às transições:

$Dist \in \mathbb{N}^m \to F$ é uma função de distribuição de probabilidade associada ao tempo de cada transição, sendo que $F \leq \infty$. Esta distribuição pode ser dependente de marcação;

$Policy \in \{prd, prs\}$ define a política de memória adotada pela transição (*prd-preemptive repeat different*, valor padrão, de significado idêntico à *enabling memory policy*; *prs-preemptive* resume, corresponde a *age memory policy*);

$Concurrency \in \{ss, is\}$ é o grau de concorrência das transições, onde *ss* representa a semântica *single server*, e *is* representa a semântica *infinite server*;

$W \in \mathbb{N}^+$ é a função peso, que associa um peso (w_t) às transições imediatas e uma taxa λ_t às transições temporizadas.

A adição de transições temporizadas introduz o conceito de habilitação múltipla, que deve ser considerado em transições temporizadas com grau de habilitação maior que um. Nesse caso a semântica de disparo deverá levar em conta a quantidade de *tokens* que podem ser disparados paralelamente. As possibilidades de semântica são:

- *Single Server - SS*: O tempo de disparo é contado quando a transição é habilitada, após o disparo da transição um novo tempo será contado se a transição ainda estiver habilitada. Portanto, os disparos ocorrerão em série, independente do grau de habilitação da transição;

- *Infinite Server - IS*: Todo conjunto de *tokens* da transição habilitada é processado simultaneamente. Então todos os *tokens* serão processados em paralelo;
- *Multiple Server*: Todo o conjunto de *tokens* será processado em paralelo até o máximo grau de paralelismo (k) definido para essa semântica.

Devida à propriedade *memoryless* "perda de memória", os modelos SPN com quantidades de lugares e transições finitas são isomórficos aos modelos CTMC, portanto podem ser traduzidos em CTMCs e resolvidos numericamente para obtenção das métricas de desempenho. Esse método de solução traz resultados bastante precisos, porém nem sempre pode ser aplicado. Uma das limitações é que a distribuição associada às transições temporizadas deve ser exponencial. Que pode ser contornada através de técnicas de *Moment Maching*, criando novas transições e lugares. No entanto, essa solução pode contribuir para a outra restrição, que é a explosão do espaço de estados da CTMC gerada pela SPN, tornando o tempo para computação das métricas proibitivo (TUFFIN et al., 2007).

Por outro lado, as métricas também podem ser obtidas por técnicas de simulação, uma alternativa quando alguma das restrições acima não for satisfeita. Os métodos de simulação permitem mais facilmente a utilização de outras distribuições para as transições temporizadas (normalmente representadas como um retângulo cinza) e por não precisarem gerar a CTMC. No entanto, em um modelo SPN que representa adequadamente um sistema real, os resultados das métricas obtidas por análise numérica podem ser mais precisos que os obtidos por simulação (considerando a simulação e análise numérica de um mesmo modelo SPN que represente fielmente o sistema), pois, na simulação os resultados são apresentadas dentro de um intervalo de confiança, enquanto os valores obtidos por análise numérica são pontuais (TUFFIN et al., 2007).

Tanto a modelagem visual, como a obtenção das métricas por análise numérica e análise estacionária podem ser realizadas por ferramentas de apoio a modelagem como o Mercury (SILVA et al., 2015). Que é capaz de construir o modelo de Markov a partir da SPN, evitando possíveis erros da construção manual.

Outros modelos como CTMC também poderiam ser utilizados, mas a grande quantidade de estados torna a tarefa impraticável para esse tipo de sistema, e modelos em Redes de Filas (*Queuing Network - QN*) (KLEINROCK, 1975; BOLCH et al., 2006) apresentam dificuldades para modelagem de sincronização e compartilhamento de recursos, por outro lado as SPNs possuem uma capacidade equivalente de modelagem de filas, além de oferecerem maior poder descritivo (MARSAN, 1988).

2.3 Modelagem de Consumo Energético

Devido à computação em nuvem ter se tornado a mais importante forma de se obter recursos computacionais, os *data certers* têm se tornado cada vez maiores, o que eleva o consumo elétrico. Portanto, o consumo de energia tem se tornado uma das principais preocupações dos

data centers. Tanto academia quanto indústria têm desenvolvido técnicas para economia de energia. Os últimos desenvolvimentos incluem *hardwares* para monitoramento, soluções de *software* para agendamento consciente (GU; HUANG; JIA, 2014), e consolidação de carga de trabalho. Em ambientes virtualizados, esse último tem sua aplicação facilitada, devido à possibilidade de migração de máquinas virtuais no menor número de máquinas físicas, para em seguida desligar os servidores ociosos (GU; HUANG; JIA, 2014; LIN; LIU; WU, 2011).

Além da consolidação, a otimização do consumo elétrico utiliza informações das VMs, que são as unidades básicas para virtualização e alocação de recurso, então, diversos trabalhos buscaram estudar o consumo e a medida de potência das VMs. Esses modelos podem ser aplicados tanto para geração de algoritmos de escalonamento, quanto para identificação de custos reais para prover uma VM. VMs com as mesmas configurações e tempo de locação podem ter consumo de energia completamente diferentes. O consumo vai depender das tarefas executadas pela aplicação que executa na VM (GU; HUANG; JIA, 2014).

Entretanto, não é possível medir o consumo de uma VM diretamente, equipamentos de medição em servidores não podem ser usados para medir o consumo da VM. Para isso modelos foram propostos na literatura. Esses modelos de consumo consideram a utilização de recursos individuais pela VM e o consumo estático da máquina física hospedeira. A potência de um servidor físico que hospeda VMs é decorrente de dois componentes o P_{Static} e P_{VM}. P_{Static} é a potência fixa do servidor, independente de executar VMs ou não, sendo a potência necessária para manter o *hardware* ligado. P_{VM} é a potência dinâmica que é consumida pela VMs executando no servidor (MÖBIUS; DARGIE; SCHILL, 2014).

Supondo que haja n VMs em um servidor, cada uma será simbolizada por VM_i , $1 \leq i \leq n$, P_{VM_i} é a potência da VM i. As potências do servidor e das VMs serão dadas por:

$$P_{Total} = P_{Static} + \sum P_{VM} = P_{Static} + \sum_{i}^{n} P_{VM_i} \tag{2.2}$$

Cada P_{VM_i} poderá ser dividida pelo consumo de seus componentes individuais, tais como: CPU, memória e entrada/saída (Input Output - IO). A potência consumida por cada componente será: $P_{VM_i}^{CPU}$, $P_{VM_i}^{Mem}$, $P_{VM_i}^{IO}$, respectivamente. Então, a potência da VM_i será:

$$P_{VM_i} = P_{VM_i}^{CPU} + P_{VM_i}^{Mem} + P_{VM_i}^{IO} + e \tag{2.3}$$

Essas potências podem ser decompostas em relação à fração alocada de cada recurso do sistema. Para isso há duas principais famílias de modelos, uma mede os contadores de monitoramento de desempenho (Performance Monitoring Counter - PMC) dos componentes, e a outra a utilização dos componentes (MÖBIUS; DARGIE; SCHILL, 2014). Nós utilizaremos os modelos baseados na utilização, pois apresentam uma tradução adequada das métricas obtidas em modelos SPN, então, a potência da VM será representada através de um modelo linear ternário:

$$P_{VM_i} = \alpha \times U_{VM_i}^{CPU} + \beta \times U_{VM_i}^{Mem} + \gamma \times U_{VM_i}^{IO} + e \tag{2.4}$$

Onde $U_{VM_i}^{CPU}$, $U_{VM_i}^{Mem}$ e $U_{VM_i}^{IO}$ representam a utilização da CPU, memória e disco, respectivamente. e é o parâmetro de ajuste. α, β e γ são parâmetros que ajustarão a utilização de cada recurso com a potência utilizada para cada tipo de sistema. Combinando as Equações 2.2, 2.3 e 2.4 é possível computar a potência total de um servidor físico com base na utilização de cada componente das VMs. Supondo que existam n VMs em um servidor, cada uma simbolizada como VM_i, e pertencendo ao intervalo: $1 \leq i \leq n$, a potência do servidor será (GU; HUANG; JIA, 2014; LIN; LIU; WU, 2011; LI et al., 2012):

$$P_{Server} = \alpha \times \sum_{i}^{n} U_{VM_i}^{CPU} + \beta \times \sum_{i}^{n} U_{VM_i}^{Mem} + \gamma \times \sum_{i}^{n} U_{VM_i}^{IO} + n \times e + P_{Static} \quad (2.5)$$

Essa relação de consumo elétrico de cada componente de cada VM é retratada na Figura 2.5. As parcelas de utilização de cada recursos de cada VM (cores escuras) representarão a utilização total de recursos, que será proporcional à potência total em um instante pelo servidor. Pode ser visto também, que a utilização máxima de todos os recursos corresponderá à potência máxima do servidor. Além disso, pode-se identificar, que sempre haverá o consumo *static*, mesmo que não haja utilização de recursos.

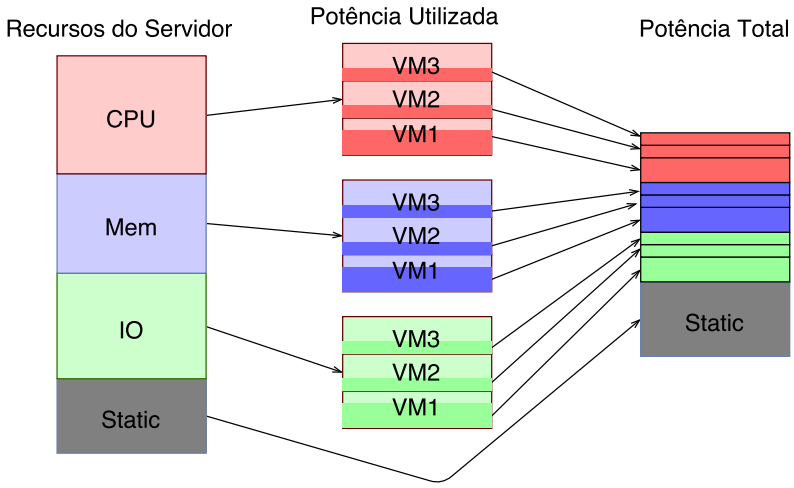
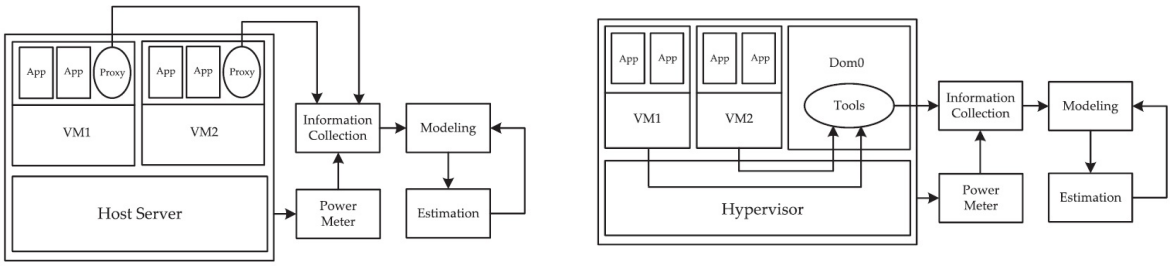


Figura 2.5: Potência de um servidor com VMs

GU; HUANG; JIA (2014) apresentam três passos para obtenção dos parâmetros necessários para criação do modelo matemático: coleta de informações; modelagem e estimação. A coleta de informações pode ser realizada por dois métodos, o caixa-branca e o caixa-preta. No método caixa-branca, um programa de monitoramento é inserido dentro de cada VM para coletar informações de utilização de cada um dos seus componentes. Este método é apresentado na Figura 2.6 (a), as medidas de utilização são obtidas de cada VM que executa um programa pré-configurado. As medições de potência são obtidas por equipamentos externos como as unidades de distribuição de potência (*Power Distributed Unit - PDU*), que podem ser facilmente acopladas e desacopladas sem afetar a operação normal do sistema.

O método caixa-preta (Figura 2.6 (b)) coleta a informação de cada VM no nível do

hospedeiro. Isto é, fora da VM, normalmente são informações providas pelo *hypervisor* que monitoram os recursos oferecidos pelo hospedeiro à VM. Esse método tem a vantagem de não gerar sobrecarga nas VMs, porém nem sempre o *hypervisor* oferece todas as informações necessárias.



(a) caixa-branca (b) caixa-preta

Figura 2.6: Metodologia de medição (GU; HUANG; JIA, 2014).

As informações coletadas são utilizadas para modelar a potência do sistema, e então, o modelo gerado é utilizado para estimar a potência de cada VM. O processo de estimar, irá validar o modelo, caso o modelo apresente um erro maior que o aceitável, serão coletados novos dados para treinar novamente o modelo.

Há diversos métodos para identificação dos parâmetros α, β e γ e "e", entre eles: modelos lineares (LI et al., 2012; BOHRA; CHAUDHARY, 2010; KANSAL et al., 2010), modelos polinomiais (VERSICK; WASSMANN; TAVANGARIAN, 2013), aprendizagem de máquina YANG et al. (2014), entre outros. Porém os modelos lineares são os mais utilizados devido à sua simplicidade de implementação, baixa sobrecarga para utilização e boa precisão, 96% no caso de LI et al. (2012). Nos métodos lineares o mais frequentemente utilizado é o dos mínimos quadrados com regressão linear múltipla (GU; HUANG; JIA, 2014; MÖBIUS; DARGIE; SCHILL, 2014). Este último será adotado neste trabalho.

2.4 GRASP

Otimização é o processo de tentar encontrar a melhor solução possível dentre todas as disponíveis. A tarefa de otimização é modelar o problema na forma de alguma função avaliável, que possa ser comparada com outra solução dada. E então, aplicar algum algoritmo buscando minimizar ou maximizar essa função objetivo. Esse processo não garante encontrar a melhor solução, pois em muitos casos o espaço de soluções é muito grande para se realizar uma busca exaustiva, tornando impossível a certeza da melhor solução (BURKE; KENDALL et al., 2005).

Dentre as estratégias de identificar soluções otimizadas, as metaheurísticas estão entre as mais efetivas técnicas de solução de problemas de otimização combinatorial e têm sido aplicadas amplamente na academia e na indústria, pois coordenam a interação entre procedimentos de buscas locais e estratégias de alto nível para criar um processo capaz de escapar de mínimos locais e realizar uma busca robusta no espaço de soluções (GENDREAU; POTVIN, 2010). Esse

tipo de método de otimização produz soluções de boa qualidade para problemas difíceis de otimização combinatorial. Como exemplos de metaheurística temos: GRASP, *Busca Tabu*, *scatter search*, *variable neighborhood search* e *Simulated Annealing* (GENDREAU; POTVIN, 2010).

A metaheurística adotada nesse trabalho para busca de soluções de boa qualidade para os modelos foi a GRASP, devido a ser facilmente implementada com poucos ajustes nos parâmetros do método, além da sua facilidade de implementação em paralelo, pois seu processo iterativo básico não possui memória. Outra vantagem é sua fácil adaptação para as características de diversos problemas (FESTA; RESENDE, 2002; BURKE; KENDALL et al., 2005).

A Grasp foi apresentada em 1989 em (FEO; RESENDE, 1989), esta é uma meta-heurística iterativa, semi-gulosa, possuindo também aleatoriedade. Na GRAPS cada iteração consiste de duas fases, a construção e a busca local. A construção, cria uma solução aleatoriamente, se a solução não for aceitável, poderá ser aplicada um reparo. Durante a busca local será investigada a vizinhança da solução gerada na construção até encontrar um mínimo local.

A GRASP em alto nível é apresentada em pseudo código no Algoritmo 1. Esse algoritmo recebe como parâmetros a quantidade de iterações (*MaxIterations*) e a semente (*Seed*) para geração aleatória, a primeira solução é definida como infinita na inicialização do algoritmo, garantindo que a primeira solução encontrada substitua a primeira. Iterativamente serão construídas soluções gulosas aleatórias iniciais que serão refinadas pela busca local. Como nesse estudo desejamos minimizar o custo, se a solução encontrada tiver o menor custo que a menor anterior *BestSolution* ela será substituída na linha 5 do Algoritmo ($UpdateSolution(Solution, BestSolution)$) (FEO; RESENDE, 1989; BURKE; KENDALL et al., 2005).

Algoritmo 1: GRASP

Input: MaxIterations,Seed

1 $BestSolution \leftarrow \infty$;
2 **for** $i \leftarrow 1$ **to** $MaxIterations$ **do**
3 $\quad Solution \leftarrow GreedyRandomizedConstruction(Seed)$;
4 $\quad Solution \leftarrow LocalSearch(Solution)$;
5 $\quad UpdateSolution(Solution, BestSolution)$;
6 **end**

Result: *BestSolution*

2.4.1 Fase de Construção

A GRASP diferencia-se de um simples algoritmo guloso com a introdução da aleatoriedade na construção dos elementos que compõem a solução. Em um algoritmo puramente guloso, a seleção do próximo elemento a ser incorporado é determinada com a

avaliação dos candidatos pela função gulosa, essa função normalmente representa uma variação na função de custo da solução inicial devido à adição de um elemento à solução parcial. O critério guloso estabelece que sempre o elemento com menor custo deva ser adicionado à solução (FEO; RESENDE, 1989; GENDREAU; POTVIN, 2010).

Algoritmo 2: Greedy Randomized Construction

Input: Seed

```
1  Solution ← 0;
2  Initialize the set of candidate elements;
3  Evaluate the incremental costs of the candidate elements;
4  while there exists at least one candidate element do
5      Build the restricted candidate list (RCL);
6      Select an element s from the RCL at random;
7      Solution ← Solution ∪ {s});
8      Update the set of candidate elements;
9      Reevaluate the incremental costs;
10 end
```

Result: *Solution*

Já na GRASP, a aleatoriedade permite uma maior variedade das soluções geradas, criando diferentes trajetórias de busca a serem seguidas mesmo que a solução inicial seja a mesma. Ele parte do mesmo princípio de algoritmos gulosos, porém com a adição da aleatoriedade. Como pode ser visto no Algoritmo 2, a cada iteração um conjunto é formado por todos os elementos que podem fazer parte da solução sem destruir sua validade. Todos os elementos são avaliados de acordo com a função de avaliação. Então, é criada uma lista restrita dos melhores candidatos (*Restricted Candidate List - RCL*), ou seja, aqueles que incorporados à solução parcial apresentam o menor custo incremental. Por fim, o elemento a ser incorporado será escolhido aleatoriamente da RCL, uma vez incorporado à solução parcial, o conjunto de candidatos é atualizado e a solução parcial é atualizada. É importante notar que a RCL é o aspecto probabilístico da heurística (FEO; RESENDE, 1989; GENDREAU; POTVIN, 2010).

Um dos parâmetros configuráveis para melhoria das soluções encontradas é relacionada aos elementos da RCL. Nós definiremos $c(e)$ o custo incremental associado ao elemento $e \in E$, sendo E o conjunto de elementos finitos que podem ser escolhidos para compor as soluções. Temos também c^{max} e c^{min} como sendo o maior e o menor custo incremental, respectivamente. A RCL é composta de elementos $e \in E$ com os menores custo incrementais c(e). A lista pode ser limitada pelo número de candidatos ou pela qualidade dos elementos. No primeiro caso a lista é composta por p elementos com os menores custos. Sendo possível também associado a um parâmetro de limite $\alpha \in [0,1]$. Que escolhe os elementos de qualidade superior a um limite, então $c(e) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$. No caso de $\alpha = 0$, corresponde a um algoritmo puramente guloso. Enquanto que $\alpha = 1$ é um algoritmo completamente aleatório (GENDREAU;

POTVIN, 2010).

A GRASP pode ser vista como uma técnica de amostragem repetitiva. Cada iteração produz amostras cujas variações dependem da natureza restritiva da RCL. Para uma RCL restrita a um único elemento, todas as iterações produzirão sempre o mesmo resultado, podendo ficar restrito a um mínimo local. Se a RCL permite muitos elementos então, haverá uma maior variação das soluções produzidas (GENDREAU; POTVIN, 2010).

Foram propostas diversas variações da fase de construção do GRASP. (MATOS; MACIEL; SILVA, 2016) abordaram a utilização da análise de sensibilidade para reduzir o tempo necessário para encontrar boas soluções. (PRAIS; RIBEIRO, 1985) variou o parâmetro α, onde o parâmetro é auto ajustado de acordo com a qualidade das soluções encontradas. (COLMENAR et al., 2016) procurou aumentar a variabilidade das soluções utilizando apenas parte dos elementos E que constituem a solução. Essas adaptações demonstram a fácil adaptabilidade da metaheuristica para cada problema. Nós utilizaremos a abordagem proposta por (COLMENAR et al., 2016) tanto para aumentar a variabilidade quanto reduzir o tempo para computação de cada construção.

2.4.2 Fase de Busca local

A solução gerada na construção não necessariamente será ótima, mesmo localmente. A fase de busca local busca mínimos locais interativamente. Ela sucessivamente troca a solução anterior por uma de menor custo local. A velocidade e efetividade da busca local depende de diversos aspectos, tais como a natureza do problema, a estrutura da vizinhança e a técnica de busca utilizada. Esta fase pode ser muito beneficiada pela qualidade da solução encontrada na fase de construção (GENDREAU; POTVIN, 2010).

Algoritmo 3: Simple Local Search

Input: Solution

1 **while** *Solution is not locally optimal* **do**

2 Find s' $\in N(Solution)$ with c(s') < c(Solution);

3 $Solution \leftarrow s'$;

4 **end**

Result: *Solution*

Normalmente são utilizadas simples buscas na vizinhança seguindo a estratégia de melhor melhoria ou primeira melhoria. No caso da melhor melhoria, toda a vizinhança é investigada e a solução anterior é trocada pelo mínimo local. No caso da primeira melhoria, a solução será a primeira de menor custo que a recebida da fase de construção. O algoritmo de melhor melhoria é apresentado em pseudo código no Algoritmo 3 (GENDREAU; POTVIN, 2010).

Algoritmo 4: VND

Input: Solution,K_{max}

$k \leftarrow 1$;
repeat
  Find s' $\in N(Solution)$ with c(s') < c(Solution);
  if *s'*<*Solution* **then**
    $Solution \leftarrow s'$;
    $k \leftarrow 1$ (centre the search around Solution and search again in the first neighborhood);
  else
    $k \leftarrow k+1$;
  end
until $k = k_{max}$;

Result: *Solution*

Outras adaptações combinam o GRASP com outras metaheuristicas para busca local. Como a busca de vizinhança variável (Variable Neighborhood Search - VNS) e sua variação, (Variable Neighborhood Descent - VND) que foram aplicadas em diversos estudos na literatura (GENDREAU; POTVIN, 2010; BURKE; KENDALL et al., 2005; SALEHIPOUR et al., 2011; HERNÁNDEZ-PÉREZ; RODRÍGUEZ-MARTÍN; SALAZAR-GONZÁLEZ, 2009). VNS é baseado no principio de sistematicamente explorar diversas vizinhanças combinado com um movimento de perturbação (conhecido como *shaking*) para escapar de ótimos locais. O VND é uma variação do VNS no qual a fase de *shaking* é excluída, sendo usualmente determinístico. Ambas as abordagens tomam vantagem ao realizar a busca em um maior número de vizinhanças até um número máximo de buscas k. O algoritmo 4 apresenta o algoritmo VND, que também será utilizado nas buscas locais neste trabalho.

2.5 Transcodificação de Vídeo

Há uma grande variedade de dispositivos buscando compartilhar conteúdo de vídeo na Internet, cada um com capacidades e características de rede heterogêneas. Por exemplo, dispositivos gravando vídeos em alta qualidade que serão compartilhados para visualização através de *smartphones*, que em muitos casos são dispositivos de baixa capacidade, com telas pequenas, resolução limitada e restrita quantidade de *codecs* aceita. Outra restrição vem da variedade de acesso à rede (Ethernet, WIFI, 4g, 3g, entre outras), cada uma com diferentes larguras de banda, taxa de erros, retransmissões, perda de pacotes e flutuações (AHMAD et al., 2005).

Uma solução para esse problema, é codificar um mesmo vídeo em vários formatos para entregar a mídia adequada a cada um desses diferentes dispositivos, condições de rede e experiência de usuário. Estratégia utilizada pelo ABR, uma técnica amplamente utilizada

para oferecer vídeo sob demanda (Video On Demand - VoD). O ABR requer que o vídeo seja codificado em diferentes versões de qualidade permitindo que o cliente escolha a versão apropriada do vídeo para o seu dispositivo e estado atual da rede (que pode mudar durante a exibição do vídeo). O ABR também requer que o vídeo seja dividido em segmentos de 2 a 10 segundos e armazenado junto com seus metadados Media Presentation Data - MPD. (KRISHNAPPA; ZINK; SITARAMAN, 2015; STOCKHAMMER, 2011).

A Figura 2.7 apresenta um esquema do funcionamento de ABR *streaming*, o vídeo original é transcodificado em todos os formatos que serão exibidos (usualmente, em qualidade baixa, média e alta). O vídeo será, então, dividido em segmentos e gerado o MPD para que os vídeos possam ser distribuídos para os diversos dispositivos e rede.

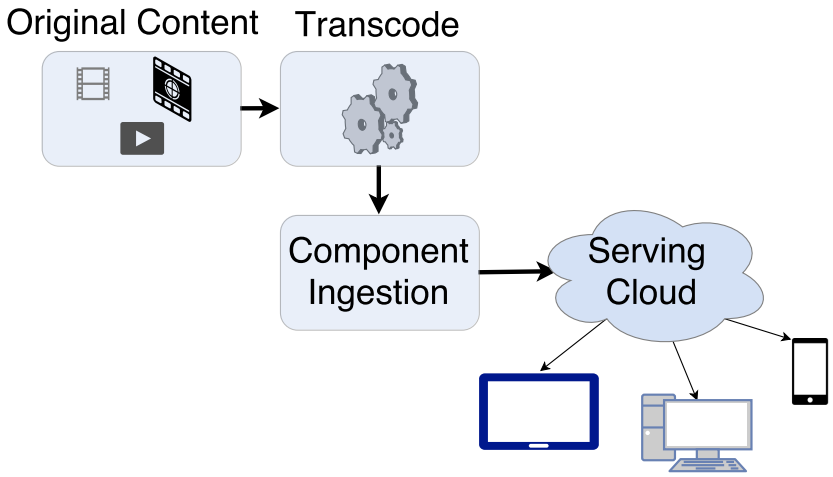


Figura 2.7: Arquitetura ABR (baseado em (STOCKHAMMER, 2011))

Transcodificação de vídeo é a atividade essencial nesse contexto. Pois é capaz de codificar um formato de vídeo em outro (também é conhecido como conversão de vídeo ou codificação). Os formatos digitais podem existir em diferentes containers, *codecs*, *bitrates*, *frame rate* e resolução de tela. Que serão descritos a seguir:

- Containers: armazenam os diversos componentes dos vídeos, imagens, sons, legendas, metadados, alguns exemplos de containers: .MOV, .FLV, .MP4, .OGG e .WMV;
- *codecs*: são a forma de compressão utilizada no áudio e vídeo. Alguns codecs de vídeo: H264, VP6 e H263. Para os de áudio: MP3, ACC e Flac;
- *bitrate*: é a taxa de dados para o vídeo exibida por unidade de tempo, normalmente medido como megabits por segundo (Mbps) ou kilobits por segundo (kbps). Essa taxa também pode ser constante *Constant Bit Rate - CBR*, e variável *Variable Bit Rate - VBR*. O VBR ajusta a taxa de dados baseado nas requisições do compressor;
- *frame rate*: É relacionada com a velocidade que o vídeo executa, uma alta *frame rate* também aumenta a *bitrate*. Essa taxa normalmente é padronizada, nos Estados

Unidos para 25 fps (metade da frequência elétrica), no Brasil utiliza-se 30 fps devido à frequência elétrica ser 60Hz;

- Resolução de tela: É o número de pixels distintos que podem ser apresentados em cada dimensão, podendo ser medido em largura $\times$ altura.

Neste trabalho utilizaremos o FFMPEG para realizar as transcodificações de vídeo. Ele é um software livre capaz de transcodificar de todas as características mencionadas acima em uma ampla variedade de formatos atuais e obsoletos. Também é capaz de executar em diversos sistemas operacionais (FFmpeg, 2016).

2.6 Trabalhos Relacionados

As pesquisas relacionadas com a área de interesse dessa dissertação abrangem temas sobre modelos de avaliação de desempenho e custo, transcodificação de vídeo, computação em nuvem, *auto-scaling*, modelos de custo elétrico em nuvem, e otimização. Alguns desses trabalhos permeiam mais de uma área, no entanto, não foram encontrados trabalhos que contenham todas as áreas cobertas por essa dissertação.

O uso de computação em nuvem para transcodificação foi abordada por LIN et al. (2013). Os autores utilizam a capacidade de paralelização oferecida por diversas instâncias na nuvem para reduzir o tempo de transcodificação de vídeos. Eles propõe um algoritmo para obter uma solução otimizada para o desempenho na transcodificação paralela de vídeos. Já em (YANG et al., 2015), a computação em nuvem foi utilizada para que fosse atingido um critério de *Quality of Service - QOS* através do provimento dinâmico de recursos da nuvem, ou seja, ajustando a quantidade de recursos dinamicamente para se adequar a variações na entrada de vídeos. Também foi utilizada uma política de que o primeiro vídeo a chegar seria o primeiro vídeo a ser transcodificado (*First come, first served - FCFS*). Entretanto, em nenhum desses trabalhos, o custo foi abordado. Esses trabalhos também trataram apenas de nuvens públicas.

Modelagem estocástica para sistemas em nuvem, foi utilizada em (DE SOUSA et al., 2014). Os autores apresentam uma estratégia para o planejamento de sistemas de ensino a distância (Virtual Learning Environment - VLE) em infraestruturas privadas. Essa estratégia visa avaliar o desempenho e custo através de modelos de Rede de Petri Estocástica (*SPN*). O modelo proposto recebe como parâmetro o tempo entre chegadas para diferentes quantidades de usuários simultâneos, e foi avaliado tempo de resposta para cada configuração de hardware e software. O modelo SPN foi validado para duas infraestruturas com grau de significância de 5%. Posteriormente o modelo foi utilizado para avaliar o tempo de resposta e custo, considerando maiores cargas de trabalho. No entanto, este trabalho não levou em conta os mecanismos de elasticidade focando no custo da aquisição do hardware necessário para implantar uma nuvem capaz de atender a demanda.

Já o mecanismo de elasticidade na nuvem foi avaliado por modelos em (CAMPOS et al., 2015). Os autores avaliam o desempenho do *auto-scaling* em nuvens privadas através de modelos de Cadeia de Markov de Tempo Contínuo (*CTMC*). Foram considerados fatores como: tipo de VM, tamanho da imagem da VM, e a probabilidade da VM estar no cache. O modelo CTMC proposto foi utilizado para identificar a influência de cada um dos fatores em relação ao tempo de instanciação de uma nova VM. Para isso foi utilizada a técnica de Planejamento de Experimentos (*Design of Experiments - DOE*).

DUPONT et al. (2015) analisou estratégias de elasticidade na nuvem. Os autores levam em conta o tempo necessário para iniciar recursos no *scaling-up*, que pode levar o sistema a não reagir a picos inesperados de carga. Também foi discutido nesse trabalho, o desperdício pela utilização parcial dos recursos alugados devido à granularidade das opções de aquisições de recursos na nuvem. Para isso os autores defenderam que a camada de software participe das ações de *auto-scaling*. Essa abordagem permitiu redefinir as configurações da elasticidade para oferecer recursos computacionais em quantidade suficiente à demanda.

Modelos também foram usados para identificar o custo decorrente da opção de diferentes tipos de contrato na nuvem. Em (RIBAS et al., 2015) os autores apresentaram uma Rede de Petri Colorida (*Colored Petri Net - CPN*) para simular a utilização de instâncias *Spot* para prover instâncias elásticas. Instâncias *Spot* não possuem um preço fixo por hora, o seu custo muda de acordo com a oferta e demanda da nuvem pública. O cliente da nuvem oferece um preço da instância, se o preço oferecido pelo provedor for inferior, a instância será criada, caso o preço seja maior que a oferta do cliente, a instância é terminada. Os autores identificaram que instâncias *Spot* podem ajudar a reduzir o custo quando comparadas com instâncias sob demanda e reservadas, porém, esse trabalho não leva em conta requisições de usuários e SLAs, que definem a demanda da aplicação por mais recursos.

Com enfoque em nuvens privadas, redução de consumo elétrico e computação verde, (BOHRA; CHAUDHARY, 2010) propôs um modelo para predizer instantaneamente a potência elétrica de máquinas virtuais hospedadas em um *hardware*. O modelo estatístico leva em conta o relacionamento entre a potência e a utilização dos subcomponentes do sistema. Então, utilizou-se técnicas de regressão linear para predizer o consumo do *hardware* e da VMs a partir da utilização instantânea dos subcomponentes que sustentam a VM (CPU, cache, memória RAM e disco). Os parâmetros do modelo foram identificados através de método caixa branca. Isto é, foi possível medir a utilização de cada componente dentro da VM. Foram executados diversos *benchmarks* para treinar o modelo através da variação da CPU, memória e IO. O modelo elétrico gerado segundo esse método alcançou uma precisão de até 94%.

(LI et al., 2012) também propõe um modelo para identificação da potência de baixa complexidade através da utilização dos componentes de maior impacto: CPU, memória e disco da VM, levando em conta também a potência do *hardware*. O modelo proposto melhora a precisão utilizando uma regressão linear múltipla por partes, gerando mais de uma regressão, levando em conta a baixa, média e alta utilização dos recursos. Esse modelo alcançou uma

precisão de mais de 96%.

Diversos trabalhos têm enfoque na redução do consumo elétrico de uma infraestrutura, como (LIN; LIU; WU, 2011) e (XIAO; CHEN; LUO, 2014), que trataram de reduzir a potência elétrica desperdiçada pela ociosidade dos servidores. A redução do consumo foi obtida através da consolidação das VMs em um menor número de servidores, e desligando os servidores que não possuíam VMs. Esses trabalhos também levam em conta as estratégias de migração das VMs para redução da quantidade de servidores necessários pelo sistema.

Buscamos também trabalhos relacionados com a união de modelos estocásticos e mecanismos de otimização. Em (MATOS; MACIEL; SILVA, 2016) os autores modelaram uma aplicação que combina a utilização de diversos *Web Services* em CTMC. O modelo permite avaliar a confiabilidade, custo e o tempo médio de resposta desse tipo de aplicação, considerando a escolha dos diferentes provedores de *Web Service*. Portanto, uma solução otimizada deve considerar a escolha de Web Services que minimizem a não-confiabilidade considerando também o tempo de resposta. Soluções otimizadas foram encontradas através da meta-heurística *GRASP*. Além disso também foi proposta uma versão do GRASP integrada com análise de sensibilidade com objetivo de reduzir o tempo necessário para encontrar uma solução.

A Tabela 2.1 foi construída a partir do levantamento dos trabalhos relacionados. Essa tabela evidencia a relação entre a proposta dessa dissertação e os trabalhos relacionados. Note que os trabalhos de transcodificação de vídeo não relacionam o custo associado ao desempenho, apresentando como principal contribuição métodos eficientes de redução do tempo de espera pela distribuição dos trabalhos de transcodificação nas VMs na nuvem. Também não se encontrou trabalhos que relacionassem a transcodificação de vídeo na nuvem com potência elétrica. Outra característica é que a potência elétrica não foi relacionada com desempenho nos outros trabalhos relacionados, também não encontramos trabalhos que combinem modelos estocásticos juntamente com modelos elétricos, com objetivo de otimizar o consumo.

Dessa forma, a importância desse trabalho se dá ao reunir diversas características desses trabalhos, apresentando uma evolução para o planejamento de infraestruturas na nuvem abordando uma maior quantidade de características oferecidas pelos mecanismos de elasticidade e ajuste dinâmico do tamanho da infraestrutura ativa. Durante esse trabalho serão apresentados modelos SPN que representem o comportamento de desempenho de sistemas de transcodificação em nuvem, levando em conta os aspectos de *auto-scaling*. Os modelos também serão utilizados para encontrar o custo pelo aluguel de nuvens públicas e o consumo elétrico em nuvens privadas. Por fim, os modelos serão utilizados para identificar configurações otimizadas para atender diferentes cargas de trabalho restringidas por requisitos mínimos de desempenho.

2.7 Considerações Finais

Neste capítulo nós apresentamos os conceitos fundamentais necessários para a compreensão deste trabalho. Inicialmente apresentamos a técnica de *auto-scaling* em nuvem.

Tabela 2.1: Relação entre a proposta desta dissertação e outros trabalhos relacionados

	Avaliação de Desempenho	Modelos	Validação	Elasticidade	Transcodificação de Vídeo	Custo em Nuvem Pública	Potência em Nuvem Privada	Otimização
Esta dissertação	✔	✔	✔	✔	✔	✔	✔	✔
(LIN et al., 2013)	✔				✔			✔
(YANG et al., 2015)	✔			✔	✔			✔
(DE SOUSA et al., 2014)	✔	✔	✔					
(CAMPOS et al., 2015)	✔	✔	✔	✔				
(DUPONT et al., 2015)	✔			✔		✔		
(RIBAS et al., 2015)		✔	✔	✔		✔		
(BOHRA; CHAUDHARY, 2010)		✔	✔				✔	
(LI et al., 2012; XIAO; CHEN; LUO, 2014)		✔	✔				✔	
(LIN; LIU; WU, 2011)		✔		✔			✔	
(MATOS; MACIEL; SILVA, 2016)	✔	✔						✔

Descrevemos quais parâmetros podem ser configurados e sua relação com o ajuste dinâmico da infraestrutura.

Em seguida abordamos as CTMCs, que são essenciais para a computação das métricas em estado estacionário dos modelos SPN, que efetivamente serão utilizados para modelar as arquiteturas apresentadas no Capítulo 3. Também apresentamos os métodos utilizados para modelagem do consumo energético que serão utilizados juntamente com a modelagem estocástica no modelo de nuvem privada da Seção 4.2.

Além disso, descrevemos a metaheurística GRASP, que será utilizada em conjunto aos modelos para identificar quais valores devem ser inseridos na configuração do sistema para que o custo seja minimizado e o SLA cumprido. Descrevemos também as características existentes nos diversos formatos de vídeo, bem como sua utilização nos sistemas de vídeo adaptativo. Por fim, abordamos os trabalhos relacionados com essa dissertação.

3

Metodologia e Arquiteturas de Transcodificação de Vídeo na Nuvem

Este capítulo apresenta a metodologia utilizada para configuração de sistemas de transcodificação de vídeo em nuvens públicas e privadas. Primeiro, será apresentada a metodologia utilizada para modelar os sistemas de transcodificação de vídeo. Descreveremos quais as principais atividades, ferramentas e técnicas para guiar o uso do modelo proposto. Também discutiremos algumas aplicações do modelo, juntamente com os passos necessários para otimização.

Em seguida apresentaremos as arquiteturas básicas de sistemas em nuvens públicas e privadas, onde serão evidenciados os diferentes graus de abstração, que apresentam diferentes graus de complexidade de modelagem e avaliação para atingir o desempenho pretendido ao menor custo.

3.1 Metodologia de avaliação

Nessa seção será apresentada a metodologia para o planejamento de infraestruturas de transcodificação com suporte ao *auto-scaling* reativo em nuvens públicas e privadas. A metodologia utilizada para obtenção dos resultados neste trabalho consiste em: entendimento da aplicação; escolha do modelo de nuvem; definição dos parâmetros; geração do modelo de custo e desempenho; validação; seleção de cenários; processo de otimização; variação de configurações; e análise dos resultados. A Figura 3.1 apresenta um modelo de alto nível baseado em UML (*Unified Modeling Language - UML*).

Após a validação, a variação da metodologia apresentada na Figura 3.2 pode ser utilizada para planejamento da configuração do *auto-scaling* em nuvens para transcodificação pelos administradores de sistemas. Essa metodologia auxilia o projetista a utilizar os modelos para cenários distintos dos apresentados neste trabalho, por exemplo: com o uso de diferentes tipos de VMs, com maior ou menor poder de processamento; nuvens com preços e modelos de custo diversos; variadas cargas de trabalho, inclusive considerando *workloads* sintéticos não

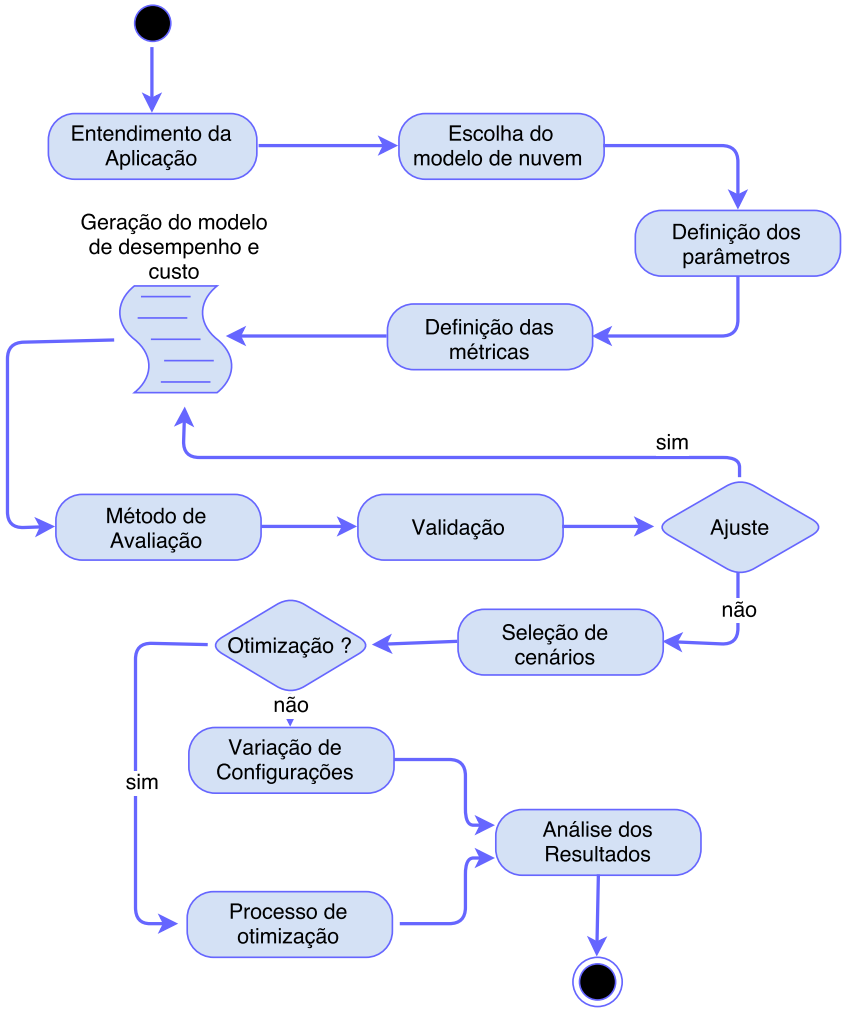


Figura 3.1: Metodologia da dissertação

exponenciais; utilização de nós físicos de diferentes capacidades de processamento e memória; infraestruturas privadas com diferentes potências elétricas; entre outras.

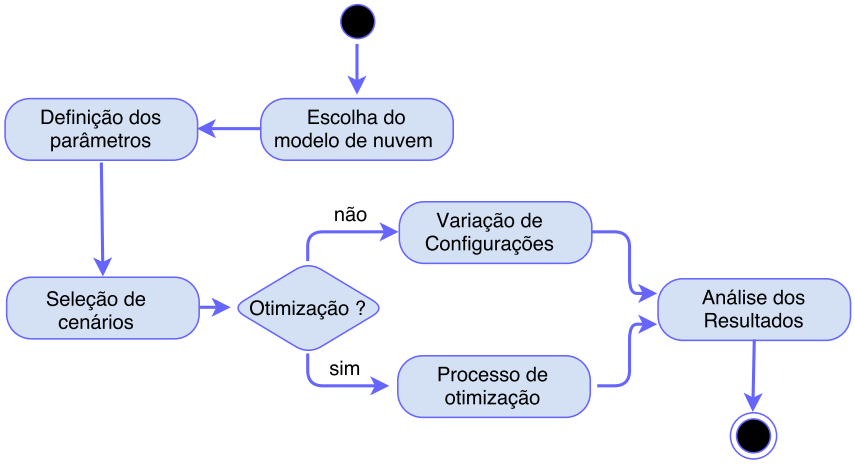


Figura 3.2: Metodologia de planejamento

A seguir descreveremos as atividades sequenciadas no diagrama retratado na Figura 3.1:

- **Entendimento da aplicação:** Essa é a primeira fase, ela busca o compreender o comportamento da utilização de recursos da aplicação que será hospedada na nuvem. Como explicado na Seção 2.1, a escolha dos recursos que serão limiares de instanciação e destruição do *auto-scaling* são específicos de cada aplicação, pois cada *software* demanda volumes diferentes de cada recurso da infraestrutura. Como exemplo, uma aplicação onde a quantidade de usuários simultâneos seja intensiva na utilização da memória e não possua demanda significante de processamento, pode ter como escolha adequada para limiar, a memória utilizada, portanto, a mudança da quantidade de memória utilizada pelas instâncias irá adicionar ou destruir VMs do sistema.

  Para compreendermos a utilização dos recursos, e identificarmos um recurso limiar adequado em uma aplicação de transcodificação, foi adicionado um *software* de monitoramento de recursos em uma VM transcodificadora, que registrou a utilização dos recursos enquanto a VM foi submetida a diferentes quantidades de requisições simultâneas. Nós utilizamos o *Apache JMeter* (JMETER, 2016) para gerar diferentes cargas, enquanto monitoramos a aplicação na VM através da ferramenta (*Nigel's performance Monitor - nmon*) (NMON FOR LINUX, 2016). Essa ferramenta de monitoramento permite capturar e armazenar a informação de diversos recursos da instância. Inicialmente nos monitoramos: utilização de cada núcleo de CPU; placas de rede (Network Interface Card - NIC); memória; discos; uso de recursos por processos; uso de *Network File System - NFS*; e sistemas de arquivos. Além disso, monitoramos a quantidade de transcodificações realizadas simultaneamente pela VM, e a influência do tamanho da fila no tempo de resposta. Algumas dessas informações também podem ser coletadas pelos sistemas de logs da nuvem, ou através de APIs diretamente do *hypervisor*.

  Os dados coletados foram analisados, e o fim dessa fase deve ser concluída com a identificação dos recursos que serão monitorados e utilizados para os limiares de instanciação e destruição no *auto-scaling*. Nossas análises para o sistema de transcodificação de vídeo permitiu concluir que o tamanho da fila é um parâmetro apropriado para os limiares utilizados, como será mostrado detalhadamente na seção 3.2. É importante ressaltar que não é necessário implantar completamente o sistema, podem ser utilizados os subcomponentes da aplicação que demandam mais recursos.

- **Escolha do modelo de nuvem:** Nessa fase identificaremos se a aplicação será hospedada em uma infraestrutura mantida pelo gestor da aplicação (nuvem privada) ou uma infraestrutura alugada de terceiros (nuvem pública). A distinção dessas duas abordagens trará relevantes questões sobre a origem dos custos e o nível de configuração que poderá ser aplicado no *auto-scaling*.

  Em nuvens públicas o custo será dado pelo tipo, contratos e tempo de locação de

máquinas virtuais. O tipo definirá os recursos computacionais de cada VM. VMs com maior capacidade de processamento ou memória, possuirão maiores custos, por outro lado responderão mais rapidamente as requisições. Os contratos serão responsáveis pela unidade de tempo para cobrança, VMs reservadas são contratadas com antecedência em contratos de longa duração (normalmente maior que um ano) mesmo que não executem qualquer atividade computacional. As VMs sob demanda podem ser adicionadas e retiradas da infraestrutura a qualquer momento, sendo pagas por hora de uso. Uma das vantagens do uso de nuvens públicas é que gestor normalmente não precisa se preocupar com a capacidade máxima da infraestrutura, pois aparenta ser infinita, além disso, também excluí a necessidade de aquisição e configuração de *hardwares*. Portanto, também reduz-se a quantidade de parâmetros a serem configurados.

Por outro lado, a escolha por infraestruturas privadas, traz maior variedade de considerações e parâmetros a serem levados em conta. O custo terá origem na aquisição do *hardware*, manutenção, no custo elétrico de manter o *data center*, entre outros. Este trabalho abordará apenas consumo elétrico como fonte de custos em nuvem privadas. Como demonstrado na Seção 2.3, o consumo elétrico é dependente tanto da utilização dos recursos de cada VM, quanto da quantidade de VMs e servidores ligados. Então, além de reduzir a quantidade de VMs permanentemente instanciadas, também é necessário reduzir a quantidade de servidores ligados. Além disso, o *hardware* utilizado irá influenciar no consumo elétrico da nuvem, pois, servidores de maiores capacidades permitem maior consolidação de recursos, ou seja, mais VMs podem permanecer a um único servidor ligado, o que reduz o consumo para manter os *hardwares* operando. Essas características irão incluir outras estratégias de economia de custos, como: definir quantidade de servidores ligados; e os limiares para ligar e desligar um servidor. Esse tipo de nuvem também requer modelos mais complexos, envolvendo o uso de equipamentos de medição de potência e consumo elétrico dos servidores.

- **Definição dos parâmetros:** Os parâmetros escolhidos definirão o nível de abstração da modelagem e irão depender do tipo de nuvem utilizado. Neste trabalho serão divididos em duas categorias, os que requerem medição e os de configuração. Os de medições, representam as características específicas da nuvem e VM utilizada. Cada tipo de VM, trará um tempo associado à duração necessária para transcodificar um vídeo, juntamente com o tempo de instanciação para que a VM se torne operacional. Em nuvens privadas, adicionalmente serão necessários os tempos para ligar e desligar um servidor. Além de ser necessário identificar os parâmetros das equações de consumo elétrico para a infraestrutura e os tipos de VM e utilizados.

  Nós também utilizamos os *softwares JMeter* e nmon para identificar os valores dos

parâmetros temporais que necessitam de medição. No caso das medições elétricas utilizamos o PDU *Watts Up* (WATTS UP, 2016). Esse equipamento permite coletar de informações da potência elétrica com uma boa precisão, dentro dos intervalos de tempo necessários para identificação dos pesos das equações.

Já os parâmetros que não requerem medições, como os limiares de instanciação e destruição, apenas requerem escolhas válidas, por exemplo, o limiar de destruição deve sempre ser menor que o de instanciação, caso contrário o sistema nunca destruirá as VMs sob demanda criadas.

- **Definição das métricas:** As principais respostas dessa análise são obtidas através das métricas. Nesse estudo buscamos identificar métricas de desempenho e de custo. As métricas de desempenho escolhidas são o tempo médio de resposta e a vazão, adicionalmente pode-se alterar o modelo para obter outras métricas customizadas, como a probabilidade de completar um ou vários trabalhos em determinado tempo, como será mostrado no Estudo de Caso 5.2.

  O custo em nuvens públicas será computado pela soma de duas métricas, a utilização de VMs sob demanda e a utilização de VMs reservadas, ambas por tipo de VM. Cabe ressaltar que esse custo irá depender de cada fornecedor. Em nuvens privadas a métrica de custo será a soma dos custos elétricos do processamento de cada VM, e o consumo elétrico estático dos servidores físicos ligados e desligados.

- **Geração do modelo de desempenho e custo:** Com a identificação dos parâmetros e métricas desejadas, o modelo pode ser gerado. O modelo representa as requisições de transcodificação de vídeo recebidas até sua codificação no formato especificado. Além de incluir o mecanismo de *auto-scaling* tanto para as VMs, quanto para os servidores em nuvens privadas. Foi adotada a modelagem estocástica por SPN, que são modelos baseados em estado que permitem tanto a análise numérica quanto simulação. Esse tipo de modelo também permite complexas configurações lógicas através de regras de habilitação em transição e pesos em arcos. Outra vantagem dessa abordagem é sua utilização de modelos com filas. Essa modelagem também permite a visualização do fluxo do sistema, auxiliando a modelagem e adaptação do modelo a outras regras. O ambiente de modelagem conta com ferramentas visuais que auxiliam o desenvolvimento e permitem a computação da métricas, dentre algumas ferramentas podemos citar o Mercury (SILVA et al., 2015) e Timenet (TRIVEDI, 2008).

- **Método de avaliação:** Após a geração do modelo, será escolhido o método de avaliação. O modelo SPN permite tanto análise numérica quanto simulação. Na análise numérica, será gerado o modelo CTMC a partir do modelo SPN, esse modelo será analisado numericamente para encontrar as métricas no estado estacionário.

Como comentado na Seção 2.2.3, esse método apresenta uma boa precisão em relação a obter os valores das métricas por simulação (quando comparados com os mesmos modelos em SPN), no entanto, nem sempre pode ser utilizado, pois o modelo pode possuir um número muito alto de estados, impedindo sua análise estacionária, ou as distribuições dos tempos associados às transições obtidas na fase de definição dos parâmetros podem não ser exponenciais (TUFFIN et al., 2007). Nesse último caso ainda é possível tentar utilizar técnicas como *moment matching*, que em alguns casos pode também gerar explosão de estados. Já a simulação, não sofre nem com a explosão de estados, nem com a obrigatoriedade do uso de distribuições exponenciais. No entanto, essa abordagem não apresenta a mesma precisão da análise numérica para o mesmo modelo SPN simulado.

Para auxiliar na decisão de qual técnica utilizar, apresentamos o diagrama de atividades na Figura 3.3. Após as medições e análise de dados, se as distribuições dos tempos das atividades medidas no sistema que serão representadas pelas transições forem exponenciais, e também não houver explosão do espaço de estados, é recomendada a análise numérica por apresentar resultados pontuais. Caso o uso de distribuições exponenciais não apresente resultados satisfatórios, o projetista é levado a escolher entre realizar diretamente a simulação ou utilizar o *moment matching*. A escolha do *moment matching* irá representar a distribuição não exponencial como uma poliexponencial. A ferramenta *Mercury* pode auxiliar nessa etapa, apresentando entre 5 diferentes distribuições poliexponenciais (Erlang, hiper-exponencial, hipo-exponencial, cox1 e cox2). Após a identificação da poliexponencial adequada para a distribuição não exponencial, o modelo deverá ser refinado para incluir o novo conjunto de transições e lugares. Posteriormente o modelo poderá ser executado por análise numérica, se não sofrer de explosão do espaço de estados.

- **Validação:** Essa fase verifica se as abstrações implementadas no modelo são consistentes com o sistema real. O sistema e o modelo devem ser executados sob a mesma carga de trabalho, e os valores das métricas do modelo e do sistema devem ser estatisticamente equivalentes, isto é, se os valores para as métricas estiverem dentro da margem de erro admitida. Para essa validação nós utilizamos tanto a comparação visual dos intervalos de confiança quanto teste t não pareado (JAIN, 1990). Para essa fase, também utilizamos o *Jmeter* para gerar a carga de trabalho e obter parte das métricas de desempenho. As métricas de custo foram obtidas no gerenciador da nuvem e também pelas medições elétricas no *Watts Up*. Caso o modelo não represente bem o sistema, ajustes devem ser realizados, que podem ser: novas medições para representar mais adequadamente as transições do sistema, ou aumentar o detalhamento da infraestrutura no modelo.

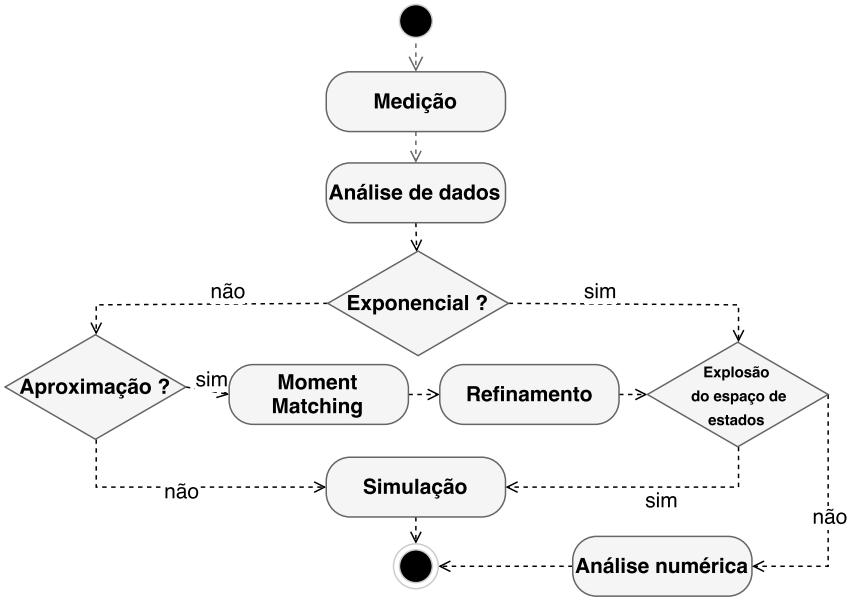


Figura 3.3: Método de avaliação do modelo

- **Seleção de cenários:** Com o modelo validado, será possível verificar o comportamento de cenários complexos, com uma boa estimativa que esse mesmo comportamento se repita no sistema real. Isso permite identificar a influência de determinado parâmetro no custo e nas métricas de desempenho. A seleção de cenários será utilizada como entrada para as fases seguintes. Nela serão escolhidas as restrições e análises desejadas, podendo procurar por soluções otimizadas ou pela variação sistemática dos parâmetros do modelo.

- **Variação de configurações:** Essa é a opção em que se procura entender o comportamento dos valores das métricas após a variação individual dos parâmetros do modelo. Tornando possível prever os valores das métricas em cenários hipotéticos de cargas de trabalho intensivas ou diferentes configurações do *auto-scaling*. Também pode ser utilizada para testar variações do modelo, como identificar a probabilidade de absorção de variadas quantidades de trabalho. Essa opção também permite análises estatísticas do modelo, como análise de sensibilidade ou probabilidade de absorção.

- **Otimizações:** Essa fase busca identificar qual a configuração de cada parâmetro para que o sistema cumpra o SLA de tempo médio de resposta e vazão com o custo otimizado. Essa técnica permite a verificação de diversos cenários otimizados, o que seria impraticável tanto no sistema real quanto na variação manual dos parâmetros do modelo, devido à complexa interação dos parâmetros. Entretanto, essa técnica também requer a medição de vários valores de transições temporizadas para cada VM analisada (tempos de transcodificações para diferentes quantidades de trabalhos simultâneos e tempo de instanciação), juntamente com medições elétricas para o caso

de infraestruturas privadas. Essa técnica não garante encontrar a melhor resposta. Mas, sendo utilizada com uma quantidade de iterações suficientemente grande, é possível encontrar resultados otimizados bons o suficiente para aplicações práticas. Cabe ressaltar que a implementação dessa técnica é viabilizada pelo uso da API dos sistemas de modelagem *SPN* através de linguagens de programação, para que seja possível a mudança automatizada dos parâmetros do modelo. A aplicação desse método pode ajudar a responder questões como:

- Como deve ser configurado o sistema em nuvem pública ou privada para que o tempo de resposta seja inferior a 45 segundos com o menor custo possível;
- Quais os custos mínimos associados à diferentes SLAs;
- Qual a infraestrutura física mínima para atender ao SLA;
- Comparações de custo entre infraestruturas privadas e públicas.

3.2 Arquiteturas

Nessa seção serão apresentadas as arquiteturas de transcodificação de vídeos em nuvens públicas e privadas, dando enfoque ao comportamento elástico dessas arquiteturas. Essa seção também apresenta o comportamento dos recursos de *hardware* virtualizado de uma VM durante a transcodificação, isso nos dará subsídios para identificar o recurso limiar para instanciação de VMs sob demanda.

3.2.1 Nuvem Pública

Um sistema de transcodificação de vídeo na nuvem, deve receber as requisições dos clientes, realizar a codificação, e armazenar os vídeos no ambiente apropriado para o compartilhamento. Esse estudo adota uma arquitetura de transcodificação similar à utilizada em (YANG et al., 2015; LIN et al., 2013). Os autores utilizaram uma VM que recebe as requisições e realiza o balanceamento de carga para as VMs que realizam a transcodificação. Se não existirem VMs disponíveis a requisição é enfileirada. Tão logo uma VM torne-se disponível ela receberá uma das requisições que estavam enfileiradas através de uma política (*FCFS*). Utilizamos também uma infraestrutura de nuvem clássica com *auto-scaling* reativo (BISWAS et al., 2015; ASSUNÇÃO et al., 2016; LORIDO-BOTRÁN; MIGUEL-ALONSO; LOZANO, 2012).

Há uma grande diversidade de provedores de infraestruturas na nuvem (*Infrastructure as a Service - IaaS*). Nesse estudo nós adotamos os preços, tipos de VMs, e contratos da Amazon (AWS, 2016), um dos maiores provedores de nuvem pública. Consideramos os quatro tipos de máquinas virtuais apresentados na Tabela 3.1, onde podem ser observadas as quantidades de

vCPUs, quantidades de memória, preços para contratos sob demanda (cobrados por hora), e reservados (cobrados por ano) de cada tipo de VM.

Tabela 3.1: Preços na nuvem Amazon Web Services EC2 (AWS, 2016)

tipo	vCPU	Mem (GB)	Reservadas ($/ano)	Sob demanda ($/h)
t2.micro	1	1	78,88	0,013
t2.small	1	2	157,68	0,026
t2.medium	2	4	315,36	0,052
t2.large	2	8	630,72	0,104

A Figura 3.4 apresenta os elementos de uma infraestrutura de transcodificação na nuvem, os usuários, as VMs reservadas (apresentadas como os blocos escuros) e VMs sob demanda (blocos claros). No estado inicial o sistema apenas possui suas VMs reservadas, um *FrontEnd* e uma VM transcodificadora. Nessa arquitetura o *FrontEnd* é responsável por encaminhar as requisições para as VMs transcodificadoras (sejam elas sob demanda ou reservadas), também irá adicionar e retirar VMs sob-demanda quando os limite pré-configurados de instanciação e destruição forem atingidos.

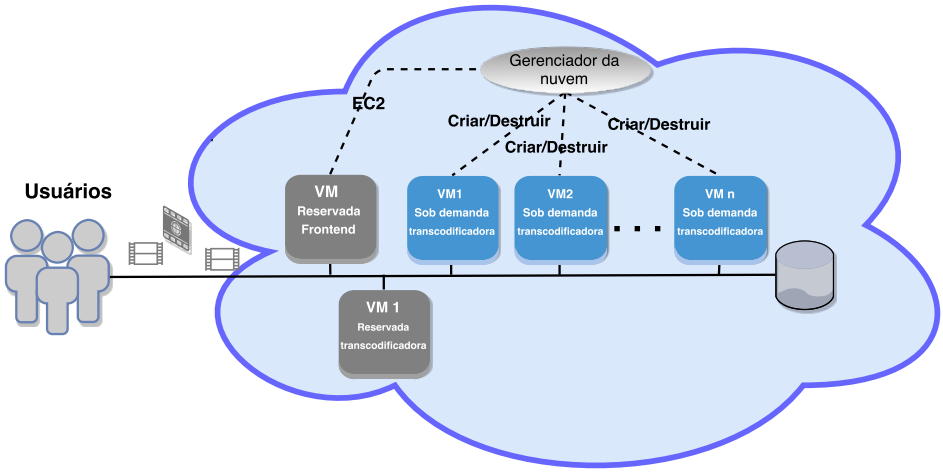


Figura 3.4: Arquitetura de transcodificação em nuvem pública

Nesse sistema, os parâmetros limites para adicionar e retirar recursos foram obtidos a partir do estudo dos recursos utilizados pela aplicação, como recomendado na Seção 3.1. Utilizamos uma instância *t2.micro* para realizar a transcodificações sequenciais de vídeos de um contêiner .AVI para o container .MP4, enquanto foram medidas as utilizações de memória e CPU, usualmente os valores desses parâmetros são utilizados para controlar a instanciação e destruição de VMs. Na Figura 3.5 vemos a utilização da CPU quando são executadas 1, 2 e 3 transcodificações simultâneas. Esse gráfico apresenta o sistema a partir da estabilidade do sistema, 300 segundos. Podemos observar que com apenas um trabalho já é suficiente para alcançar mais de 80% de uso, e para mais de uma transcodificação, toda a capacidade de

processamento é atingida.

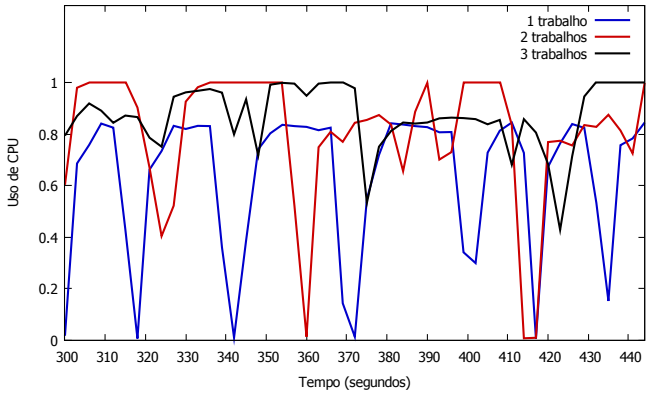


Figura 3.5: Uso de CPU na transcodificação

A memória também é quase totalmente utilizada durante a transcodificação, como observado na Figura 3.6, mantendo a utilização em torno de 94% em todos os cenários. Embora os recursos sejam quase completamente utilizados durante as execuções, as VMs completaram todas as transcodificações, apresentando apenas queda de desempenho, como visto na Tabela 3.2, o tempo para transcodificar um vídeo aumenta com aumento da quantidade configurada máxima de trabalhos simultâneos aceitos. Então, não é viável a escolha da utilização da CPU nem da memória como limiar para identificar o instante para adicionar e retirar VMs, pois suas mudanças não permitem controlar adequadamente o tempo de resposta do sistema. Por outro lado, o tamanho da fila de requisições permite controlar o *auto-scaling* em relação ao tempo médio de resposta, o quanto maior a fila existente, maior o tempo para completar a transcodificação, então será definido um tamanho de fila para adicionar ou retirar recursos, reduzindo o tempo de espera.

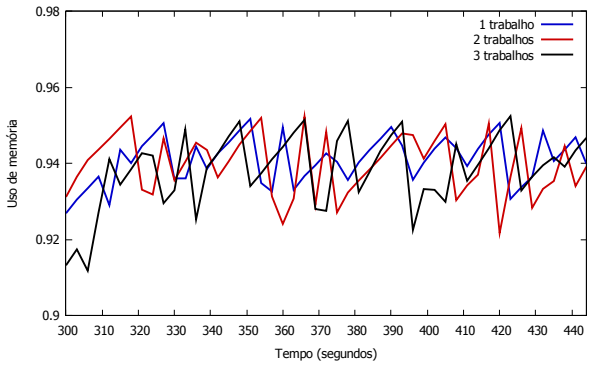


Figura 3.6: Uso de memória na transcodificação

O comportamento do sistema quando recebe requisições pode ser visto no diagrama de sequência na Figura 3.7. Ao receber uma solicitação por transcodificação, o *frontend* verifica a existência de VMs transcodificadoras que possam aceitar mais trabalhos simultâneos (condição *hasFreeTranscoder*). Caso haja um transcodificador disponível, ele será utilizado para transcodificação da requisição, caso não exista, a requisição será enfileirada (*1.2 :addQueue(request)*), e então, será verificada a necessidade da criação de uma nova VM sob

Tabela 3.2: Tempo de transcodificação da VM t2.micro

Trabalhos simultâneos por VM	Tempo (segundos)
1	20,5713
2	33,2408
3	44,0003

demanda. Essa verificação leva em conta: o atual tamanho da fila; a quantidade de transcodificadores ativos; e o limiar de instanciação na condição $[queue.size() >= (transcoders.size()) * Instthreshold)]$. Essa condição utiliza o parâmetros, limiar de instanciação (*Instthreshold*) para definir o tamanho máximo da fila por VM, menores valores dessa variável aumentará a quantidade de VMs sob demanda utilizadas para suprir a demanda, o que aumenta também o custo e reduz o tempo de resposta.

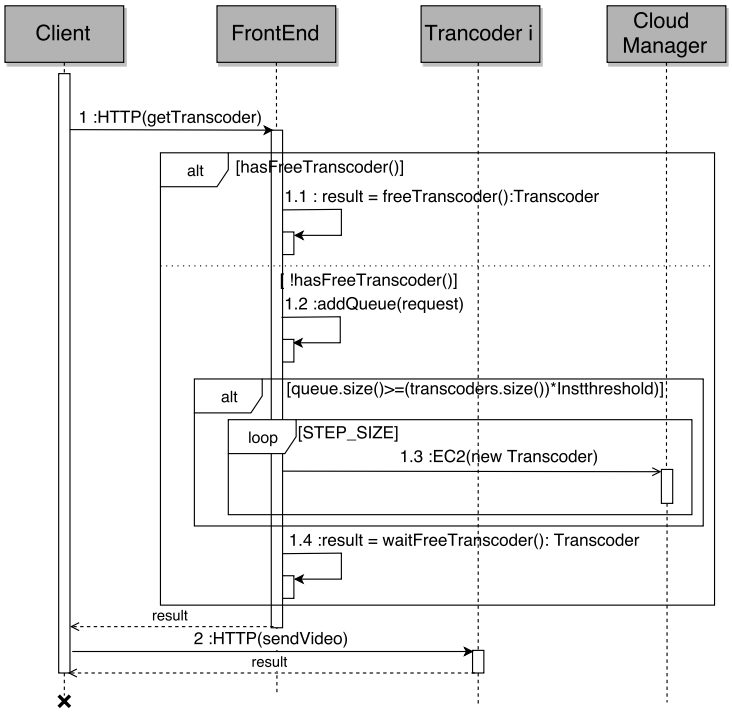


Figura 3.7: Mecanismo de processamento de requisições

Se a condição para instanciação for verdadeira, serão solicitadas tantas VMs transcodificadoras quantas forem definidas pela variável *STEP_SIZE*. Essa solicitação de VMs será feita pela solicitação assíncrona *1.3 :EC2(new Transcoder)*. Note que essa instanciação não ocorre automaticamente, leva tempo para criar uma nova VM e iniciar toda sua pilha de *software*. Após essa verificação a solicitação aguardará um transcodificador se tornar disponível pelo término de atividades ou completada a instanciação de uma VM.

O teste da destruição de VMs será realizado ao fim de cada transcodificação, onde será verificado o novo tamanho da fila e a quantidade de instâncias sob-demanda ativas. O parâmetro que definirá o momento da destruição é o limiar de destruição. É importante notar que as atividades de *scaling up*, *scaling down*, e balanceamento de carga, também podem ser realizados

pelas ferramentas já existentes na maioria dos provedores de nuvem, que podem verificar os recursos e logs da aplicação, como também, podem ser realizadas pela API EC2, como utilizado neste trabalho.

3.2.1.1 Definição dos Parâmetros

Por fim, podemos identificar nessa seção os elementos essenciais para a modelagem da arquitetura de um sistema de transcodificação na nuvem elástica, sendo eles:

- o tipo de VM: irá influenciar no tempo de transcodificação e o tempo para adição de uma nova VM sob demanda;
- a quantidade de VMs reservadas: VMs que embora sempre tenham custo mesmo quando não utilizadas, também não tomam tempo para instanciação e possuem menor custo para usos em longos períodos;
- o limiar de instanciação: define o momento da adição de novas VMs sob demanda, esse valor deve considerar que a VM não será adicionada instantaneamente, pois levará tempo para se tornar completamente operacional, dependendo do tipo de VM e nuvem utilizada;
- o limiar de destruição: define o momento para destruir VMs não necessárias para o cumprimento do SLA;
- a quantidade de trabalhos por VM: essa variável deverá levar em conta que VMs com mais trabalhos simultâneos levam mais tempo para executar, mas dependendo da quantidade de recursos e preço de uma VM, pode ser vantajoso utilizar poucas VMs maiores com mais trabalhos simultâneos que muitas VMs pequenas com trabalhos individuais;
- o step size: define a quantidade de VMs adicionadas a cada vez que o limiar de instanciação for atingido;
- carga de trabalho esperada, incluindo sua distribuição e o
- custo de cada tipo e contrato de VM.

3.2.1.2 Definição das Métricas

Um dos objetivos desse trabalho é verificar qual a configuração que deve ser definida nos parâmetros para sejam cumpridos os acordos de nível de serviço ao menor custo possível. Os SLAs foram definidos em termos de métricas de desempenho mínimos que o sistema deve servir. As métricas de desempenho utilizadas foram a vazão (*throughput*) e o tempo médio de resposta (*mean response time*).

A vazão pode ser definida como a taxa das requisições que foram servidas por unidade de tempo. Essa taxa normalmente deve crescer junto com o crescimento da taxa de chegada de trabalhos, isso deve acontecer até certo ponto. O valor máximo da vazão sob condições de cargas de trabalho ideais é chamado de *capacidade nominal* (JAIN, 1990). Neste trabalho será considerada a quantidade de vídeos transcodificados ($Transc\_jobs$) em um intervalo de tempo (T) de todas as VMs transcodificadoras, sendo calculado na Equação 3.1.

$$TP = \frac{Transc\_Jobs}{T} \tag{3.1}$$

O tempo médio de resposta pode ser obtido pela Lei de Little (LITTLE, 1961) (Equação 3.2), que relaciona o número médio de trabalhos em um sistema (N_{Jobs}), a taxa de chegada ($ARRIVAL_{rate}$) e o tempo médio de resposta (RT). A Lei de Little requer um um sistema estável, ou seja, que possua uma taxa de requisições menor que a taxa de processamento dos servidores. É importante ressaltar que no nosso sistema a taxa de chegada real não necessariamente é a taxa de chegada efetiva, pois alguns trabalhos podem se perder, ou por termos uma fila finita, serem descartados, então utilizamos a taxa de chega efetiva, como recomendado pelo autor (JAIN, 1990).

$$N_{Jobs} = ARRIVAL_{rate} \times RT \tag{3.2}$$

A Equação 3.3, apresenta o custo, como sendo a soma dos custos das VMs reservadas e sob demanda pelo período de um ano (devido ao custo dos contratos reservados serem normalmente feitos em prazos definidos nesse intervalo). As VMs sob demanda são calculadas como a soma de horas de utilização T dessas VMs pelo período de um ano. Já as VMs reservadas será pelo produto da quantidade contratada (VM_{Res}) pelo custo. Também consideraremos a utilização de um único tipo de VM, portanto teremos um único custo para cada tipo de contrato, sendo $Cost_{Res}$ o custo reservado e $Cost_{Ond}$ o custo sob demanda.

$$Cost(T) = VM_{Res} \times Cost_{Res} + T \times Cost_{Ond} \tag{3.3}$$

3.2.2 Nuvem Privada

A arquitetura da utilização de nuvens privadas dentro do contexto de redução de custos para transcodificação de vídeos apresenta mudanças significativas dos elementos apresentados em nuvens públicas. A origem dos custos não é mais consequência do aluguel da infraestrutura de terceiros, pois tem origem na aquisição de *hardware*, mão de obra, refrigeração e consumo elétrico dos servidores, sendo esse último um das maiores fontes de custo para *data centers* (WU; LIN; PENG, 2016). Outra diferença é o maior controle sobre toda a infraestrutura, em nuvens privadas tanto as VMs quanto os equipamentos de *hardware* são visíveis e passíveis de gerenciamento.

A arquitetura em alto nível da transcodificação de vídeo é apresentada na Figura 3.8.

Nela os clientes enviam os vídeos para a VM *FrontEnd* (VM cinza), que irá encaminhar a requisição para alguma das VMs transcodificadoras disponíveis, ou requisitar a criação de novas VMs, da mesma forma que em nuvens públicas. No entanto, nesse caso a capacidade será limitada pelos recursos disponíveis dos servidores físicos (em infraestruturas públicas esse limite aparenta ser infinito). A capacidade disponível na Figura 3.8 é representado pelo espaço branco acima dos *hypervisors* de cada servidor. Nessa arquitetura a VM *FrontEnd* ainda irá requisitar a instanciação e destruição de VMs transcodificadoras para o gerenciador da nuvem. Quando forem atingidos os limiares definidos pelo tamanho da fila, o gerenciador da nuvem adicionará uma VM em um dos servidores disponíveis.

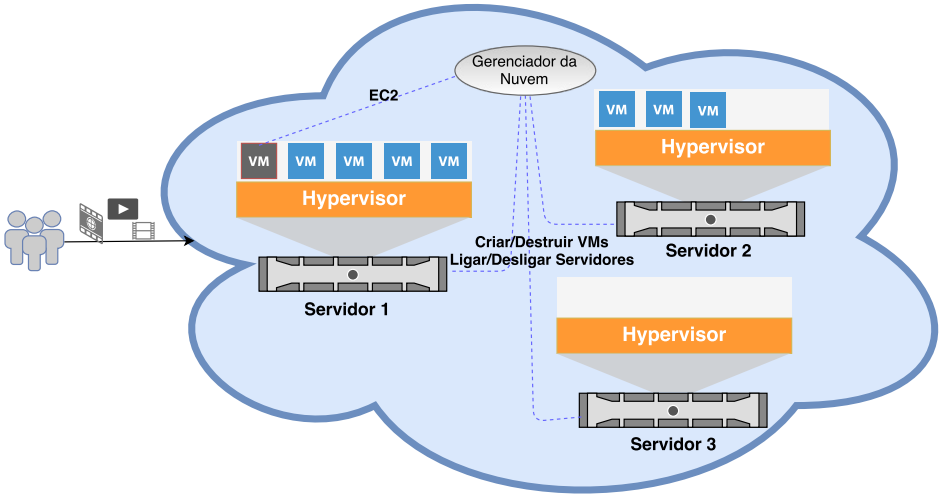


Figura 3.8: Arquitetura de transcodificação em nuvem privada

Embora não haja mais custo pela locação de VMs, ainda é importante manter um número mínimo de VMs instanciadas, pois mesmo uma VM ociosa irá consumir energia. Outra vantagem de manter um número mínimo de VMs, é a possibilidade de consolidação de servidores, a economia dessa abordagem é decorrente do custo de manter um servidor ocioso ligado, como discutido na Seção 2.3. A consolidação adapta a quantidade de servidores ativos à carga atual, técnica conhecida na literatura como data center *right-size* (LIN et al., 2013; SHARMA; SAINI, 2016; XIAO; CHEN; LUO, 2014). Essa técnica aplicada dinamicamente, liga os servidores quando houver alta carga e aciona o modo de economia de energia (desligado ou *standby*) dos servidores ociosos quando houver baixa demanda (LIN et al., 2013).

Utilizamos uma arquitetura *right-size* para o consumo elétrico dos servidores, similar à utilizada em (SHARMA; SAINI, 2016), onde os autores utilizaram limiares para ligar e desligar servidores ociosos. Eles utilizaram dados históricos para identificar os limiares utilizados. Nós utilizaremos a capacidade disponível para instanciar uma VM. No caso dos servidores da Figura 3.8, o servidor 3 poderia ser desligado dependendo do limiar de capacidade disponível mínima

configurada, e seria ligado novamente quando a capacidade disponível for igual à definida no limiar para ligar.

3.2.2.1 Definição dos Parâmetros

A arquitetura de transcodificação de vídeo em nuvem privada necessita pelo menos da configuração dos seguintes parâmetros:

- o tipo de VM, que irá influenciar no tempo de transcodificação e o tempo para adição de uma nova VM, também irá definir o consumo elétrico máximo de uma VM, quanto mais recursos físicos uma instância puder usar maior será o consumo, por outro lado, VMs com mais recursos evitam o consumo elétrico de manter diversas VMs ligadas dadas pelo *e* e o consumo elétrico da memória RAM pelo Sistema Operacional - SO, como pode ser deduzido da Equação 2.3. É importante notar que embora não existam configurações fixas de recursos de VMs nesse tipo de infraestrutura, por simplicidade, adotamos os mesmos tipos de VM da análise da nuvem pública;
- a quantidade mínima de VMs que sempre devem permanecer ligadas mesmo que não estejam processando transcodificações;
- o modelo elétrico de consumo das VMs e servidores, contendo a equação 2.5 de cada tipo de VM utilizada, e o consumo estático do servidor (P_{Static});
- o limiar de instanciação, que define o momento da adição de novas VMs, esse valor deve considerar que a VM não será adicionada instantaneamente, além disso, adicionar VMs possui um consumo elétrico. Entretanto, manter uma VM ociosa também possui um custo, levando a um conflito de escolha entre reduzir o limiar de instanciação ou manter VMs ociosas;
- o limiar de destruição: definirá o momento para destruir VMs não necessárias para o cumprimento do SLA;
- a quantidade de trabalhos por VM: Quanto maior o número de trabalhos simultâneos maior a utilização dos recursos, portanto maior consumo (Equação 2.3), também aumentará o tempo para realizar um trabalho (Tabela 3.2).
- o *step size*, definirá a quantidade de VMs adicionadas a cada vez que o limiar de instanciação for atingido;
- carga de trabalho esperada, incluindo sua distribuição;
- limiar para ligar o servidor, esse parâmetro verificará a capacidade atual de instanciação de novas VMs, e então, decidirá sobre a necessidade de mais um servidor ligado. Esse parâmetro deve considerar que ligar um servidor, requer tempo

para que esse tenha iniciado o seu SO e ser reconhecido pelo gerenciador da nuvem. Também deve levar em conta o custo elétrico para ligar um servidor;

- limiar para desligar um servidor: Verifica se há mais capacidade de instanciar novas VMs que o necessário, para então desligar um servidor. Deve considerar também o tempo e o consumo elétrico associado e
- capacidade dos servidores: A quantidade de VMs que um servidor pode manter. Essa quantidade irá variar dependendo do *hardware* de cada servidor e também do tipo de VM utilizada. É importante notar que *hardwares* com maior capacidade também permitem maior consolidação, evitando o custo associado de diversos P_{Static}, porém podem possuir maior custo de aquisição.

3.2.2.2 Definição das Métricas

As métricas de desempenho para as nuvens privadas são iguais às das nuvens públicas (Seção 3.2.1.2), a diferença é a forma de cálculo do custo, que será dado pela energia consumida pelos servidores somado com as VMs, no sistema real esse valor será obtido pelas medições através do PDU *Whatts Up*, como apresentado no Anexo A, onde será computada a potência média e multiplicada pelo intervalo de tempo T, como pode ser observado na Equação 3.4.

$$E(T) = P_{mean} \times T \tag{3.4}$$

3.3 Considerações Finais

Neste capítulo nós inicialmente apresentamos a metodologia utilizada para elaboração deste trabalho. Ela incluiu: o estudo do sistema e sua aplicação em nuvens públicas e privadas, identificação dos parâmetros e métricas; construção e validação do modelo; e otimização e análise de cenários. Também apresentamos a parcela das atividades que devem ser aplicadas pelo usuário do modelo já validado em uma infraestrutura real.

Além disso, detalhamos as arquiteturas de nuvem pública e privada, descrevendo a interação da chegada de vídeos no sistema e a instanciação e destruição de máquinas virtuais. Apresentamos também a descrição dos parâmetros e métricas para cada uma dessas arquiteturas.

4

Modelos e Otimização

Este capítulo apresenta os modelos utilizados para representar as arquiteturas de transcodificação de vídeo em nuvens públicas e privadas. Os modelos gerados são a base para verificar o comportamento do sistema em relação aos seus parâmetros. Apresentaremos também, as comparações com sistemas reais, onde os modelos foram validados com nível de confiança de 95%. Por fim, descreveremos a otimização, onde as métricas obtidas pelos modelos são os elementos chave para a avaliação da função objetivo.

O capítulo será dividido da seguinte forma: inicialmente as Seções 4.1 e 4.2 apresentam os modelos de nuvem pública e privada, respectivamente. Também são apresentadas com as métricas de desempenho e custo de cada modelo; nas seções 4.3 e 4.4 são apresentadas as validações e infraestruturas de teste usadas; a Seção 4.5 descreverá os algoritmos do GRASP combinados com a avaliação dos modelos.

4.1 Modelo Nuvem Pública

Como apresentado na Seção 3.2.1, o desempenho e custo do sistema de transcodificação na nuvem depende principalmente da configuração dos parâmetros: tipo de VM (influencia no tempo para conversão e o tempo de instanciação de VMs sob demanda); limiar de instanciação (*scaling up*); limiar de destruição (*scaling down*); *step size*; número de VMs reservadas; número máximo de trabalhos por VM; e os custos de alugueis do provedor de nuvem. Então, o administrador do sistema deve ajustar esses parâmetros adequadamente tentando encontrar o menor custo que cumpra o SLA para uma taxa de chegada de trabalhos esperada. É difícil determinar a melhor configuração apenas ajustando e verificando o efeito dessas mudanças no sistema. A modelagem pode ser útil para checar o efeito dessas mudanças antes que elas sejam implementadas.

Então, nós propomos um modelo SPN para representar esse sistema e ajudar na configuração desses parâmetros na nuvem. A Figura 4.1 apresenta esse modelo, que combina o processo de transcodificação, o mecanismo de instanciação e destruição através do *auto-scaling*. Esse modelo tem o propósito principal de calcular o tempo médio de resposta, vazão e o custo

de aluguel das VMs transcodificadoras, usando como entrada os parâmetros de configuração do sistema e o tempo entre chegadas de novas requisições.

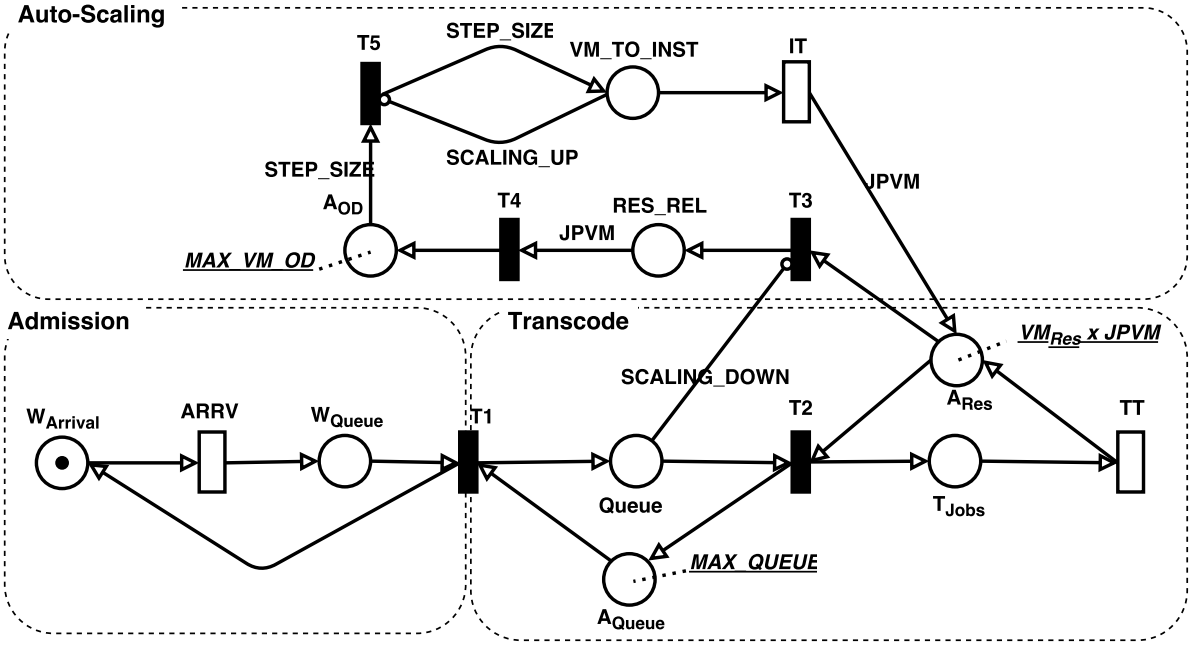


Figura 4.1: Modelo de transcodificação de vídeo em nuvem pública

Os significados dos nomes utilizados no modelo estão na Tabela 4.1, e a descrição dos atributos das transições na Tabela 4.2. Os parâmetros da distribuição das transições temporizadas devem ser providos pelo usuário do modelo, cada tipo de VM possuirá diferentes tempos para transcodificação e instanciação, que devem ser fornecidos para cada tipo de VM a ser considerado. Os valores podem ser obtidos através de medidas em um sistema já hospedado na nuvem ou considerando cargas de trabalho esperadas em diferentes VMs em um sistema não implantado. Como já explicado na Seção 3.1, a distribuição poderá influenciar no método de avaliação do modelo, podendo ser pela análise numérica ou simulação.

O modelo é dividido em 3 sub-redes: *Admission*, que trata da chegada de trabalhos; *Transcodificação*, responsável pela fila e transcodificações das requisições; e *Auto-Scaling*, que trata da adição e retirada de recursos para processamento dos trabalhos. A sub-rede de admissão é composta por dois lugares$\mathbf{W_{Arrival}}$ e $\mathbf{W_{Queue}}$, que representam a espera entre chegada de trabalhos e a aceitação desse trabalho na fila, respectivamente, e a transição **ARRV**. Os *tokens* em $\mathbf{W_{Arrival}}$ e $\mathbf{W_{Queue}}$ representam as requisições por codificação de vídeos.

Os tempos entre chegadas de trabalhos em nossa SPN é atribuído à transição **ARRV**, nós consideramos que os tempos entre disparos são exponencialmente distribuídos, essa suposição pode ser modificada, alterando essa distribuição. A sub-rede de admissão também não considera o atraso proveniente da transmissão do cliente para o sistema. A transição **ARRV** leva em conta apenas o tempo que as requisições entraram no sistema, ou seja não são levados em conta as perdas provenientes da rede.

A transição **T1** representa o recebimento da requisição, note que é uma transição imediata

Tabela 4.1: Descrição das variáveis - modelo público

Label	Description
$W_{Arrival}$	Espera pela requisição de novas transcodificações.
ARRIVAL	Tempo entre chegada de trabalhos.
W_{Queue}	Espera pela disponibilidade da fila.
Queue	Trabalhos na fila.
Max_Queue	Capacidade máxima da fila do sistema.
A_{Queue}	Capacidade disponível na fila.
A_Res	Recursos de transcodificação disponíveis.
T_Jobs	Quantidade de transcodificações sendo realizadas.
TT	Tempo para transcodificação.
JPVM	Quantidade máxima de trabalhos simultâneos por VM.
A_{OD}	Capacidade disponível de VMs sob demanda.
Scaling_UP	Condição para instanciar VMs sob demanda.
Scaling_Donw	Condição para destruir VMs sob demanda.
Step_Size	Quantidade de VMs requisitadas por *scaling up*.
VM_TO_Inst	Quantidade de VMs sendo instanciadas.
VM_{Res}	Número de VMs reservadas.
MAX_VM_OD	Número de máximo de VMs sob demanda.
THR_INST	Limiar de instanciação de VMs sob demanda.
THR_DEST	Limiar de destruição de VMs sob demanda.

Tabela 4.2: Atributos das transições - modelo público

Transition	Type	Server Semantic	Weight	Priority
ARRV	Timed	Single Server	-	-
TT	Timed	Infinite Server	-	-
IT	Timed	Infinite Server	-	-
T1	Immediate	-	1	1
T2	Immediate	-	1	1
T3	Immediate	-	1	2
T4	Immediate	-	1	1
T5	Immediate	-	1	1

(simbolizado por um retângulo preto), então, ela não possui atraso associado. Então, dispara assim que estiver habilitada pelas pré-condições, um *token* em $\mathbf{W_{Queue}}$ e pelo menos um em $\mathbf{A_{Queue}}$.

Quando **T1** dispara a sub-rede de transcodificação é alcançada, um *token* é retirado de $\mathbf{W_{Queue}}$ e $\mathbf{A_{Queue}}$ e tem como pós-condição a marcação do lugar $\mathbf{W_{Arrival}}$ (permitindo um novo disparo) e a adição de um *token* em **Queue**. A quantidade de *tokens* em **Queue** representa o enfileiramento de requisições. O enfileiramento ocorre quando não há capacidade instanciada disponível para servir a requisição recém-chegada. Se houver capacidade disponível a transição **T2** dispara e posteriormente a requisição é processada.

Uma requisição enfileirada não necessariamente representa o não cumprimento do SLA, no diagrama de sequência da Figura 3.7 representa a chamada do método (*1.4*

waitFreeTranscoder()). Os *tokens* em $\mathbf{A_{Queue}}$, representam a capacidade ainda disponível para enfileirar requisições, no início do sistema está com a capacidade máxima *MAX_QUEUE*. Os *tokens* em **Queue** também serão utilizados como condição para mudança da quantidade de recursos no modelo. Em **SCALING_UP** diretamente na condição e em **SCALING_DOWN** pela comparação do peso da condição do arco inibidor com a quantidade de *tokens* em **Queue**, essas condições definirão quando ocorrerão as instanciações e destruições de VMs.

O disparo de **T2** representa o início da transcodificação, esse disparo ocorre com a retirada de uma requisição da fila (*token* em **Queue**), e também o a utilização de capacidade das VMs disponíveis (*token* de $\mathbf{A_{Res}}$). Então, um *token* é armazenado no lugar $\mathbf{T_{Jobs}}$, para representar um vídeo em processo de transcodificação. Os *tokens* em $\mathbf{A_{Res}}$ representam a capacidade disponível de transcodificação. No início da execução esse lugar possui a quantidade de *tokens* igual ao número de VMs transcodificadoras reservadas ($\mathbf{VM_{Res}}$) multiplicado pelo número de trabalhos simultâneos que uma VM pode executar (**JPVM**). Portanto, o lugar $\mathbf{A_{Res}}$ inicia com $\mathbf{VM_{Res}} \times$ **JPVM** *tokens*.

O tempo que os trabalhos permanecem em transcodificação depende da transição **TT** (Tempo para trancodificação), Esta transição tem a semântica *infinite server*, então, cada vídeo em transcodificação é processado independentemente. É importante notar que o tempo de transcodificação depende do tipo de VM e da quantidade de trabalhos simultâneos realizados. VMs que processam múltiplos vídeos em paralelo demoram mais para transcodificar um vídeo. Também é importante observar que todas as VMs no sistema de transcodificação são homogêneas em relação à configuração, quantidade de recursos alocados e número de trabalhos simultâneos.

A instanciação de VMs sob demanda é dada pela transição **T5** na sub-rede de *Auto-Scaling*, o que representa a requisição por mais recursos. O disparo de **T5** depende da capacidade de recursos disponíveis, representada pelo número de *tokens* em $\mathbf{A_{OD}}$. Em infraestruturas públicas essa quantidade de VMs disponíveis para instanciação normalmente pode-se considerar ilimitada, mas o usuário pode definir um número máximo de VMs sob demanda, que será dado por **MAX_VM_OD**.

O disparo de **T5**, também depende da Condição 4.1, que define a multiplicidade do arco inibidor **SCALING_UP**. Devido a esse arco inibidor, a transição **T5** será habilitada apenas se o número de *tokens* em **VM_TO_INST** for menor que a multiplicidade do arco. Assim se a Condição 4.1 for verdadeira, a multiplicidade de arco será $\#\mathbf{VM\_TO\_INST} + \mathbf{1}$, o que habilita o disparo de **T5** mesmo que já existam outras VMs em instanciação (**#VM_TO_INST** significa a quantidade de *tokens* no lugar **VM_TO_INST**). Se a Condição 4.1 for falsa, a multiplicidade será zero, o que impede o disparo da transição **T5**.

Note que a Condição 4.1 depende do **THR_INST** (limiar de instanciação), que define quando uma instanciação ocorrerá. Os outros fatores para instanciação dados por essa condição levam em conta: o tamanho da fila (**Queue**); a quantidade de VMs já instanciadas, dadas pelas quantidades de *tokens* em $\mathbf{A_{Res}}$ e $\mathbf{T_{Jobs}}$; e também a quantidade de VMs requisitadas em processo de instanciação (**VM_TO_INST**). A sintaxe usada para expressar essa condição é compatível

com a ferramenta Mercury (SILVA et al., 2015).

$$
\begin{aligned}
IF(\#Queue >= ((\#VM\_TO\_INST + \\
&((\#A_{Res} + \#T_{Jobs} + \#RES\_REL)/JPVM)) \times THR\_INST)) : \\
&(\#VM\_TO\_INST + 1) \\
ELSE \\
&(0)
\end{aligned}
\tag{4.1}
$$

O número de VMs a serem instanciadas é definido pela variável **STEP_SIZE**, que é a multiplicidade do arco de **T5** até **VM_TO_INST**. A quantidade de *tokens* no lugar **VM_TO_INST** significa a quantidade de VMs esperando por instanciação, isto é, o disparo da transição **IT** (Tempo para instanciação). Cada disparo de **IT** coloca **JPVM** *tokens* no lugar $\mathbf{A_{Res}}$, representando a quantidade de trabalhos simultâneos adicionados ao sistema. **IT** deve ser medido para cada tipo de VM utilizada, e deve considerar o tempo para instanciação junto com o tempo para iniciar o SO e toda a pilha de *software* necessária para o serviço de transcodificação. **IT** é uma transição *infinite server*, correspondendo com a chamada assíncrona *1.3 EC2(new Transcoder)* no diagrama de sequência da Figura 3.7.

Por outro lado, a destruição de VMs devido à baixa demanda é representada quando **T3** dispara e os recursos associados a uma VM são depositados no lugar **RES_REL**. Esses recursos se tornaram disponíveis novamente quando **T4** disparar e colocar os **tokens** correspondentes no lugar $\mathbf{A_{OD}}$. O disparo de **T3** depende da Condição 4.2, que define a multiplicidade do arco **SCALING_DOWN**. A transição **T3** será habilitada dependendo do tamanho da fila e o número de VMs instanciadas. Quando não há VMs sob demanda instanciadas, **T3** não será habilitada. É importante notar que a destruição de VMs é relacionada com o valor atribuído à variável **THR_DEST** (limiar de destruição), e essa variável definirá juntamente com **THR_INST** o tempo que as VMs sob demanda permanecerão ativas.

$$
\begin{aligned}
IF(MAX\_VM\_OD > \#A_{OD}) : \\
&((((\#A_{Res} + \#T_{Jobs} + \#RES\_REL)/JPVM) - 1) \\
&\times THR\_DEST) \\
ELSE \\
&(0)
\end{aligned}
\tag{4.2}
$$

O modelo retratado na Figura 4.1 permite a computação de métricas como a vazão e o tempo médio de resposta. A vazão de todo o sistema, TP, pode ser calculada pela Equação 4.3, isto é, como o produto da taxa de disparo da transição **TT** com o número esperado de *tokens* no lugar $\mathbf{T_{Jobs}}$. A Equação 4.3 assume que a distribuição do tempo entre transcodificações é exponencial. Já a Equação 4.4 considera uma distribuição Erlang com o tempo médio de resposta

$\mathbf{T_{Resp}}$ e $\mathbf{k_{Resp}}$ a quantidade de fases.

$$TP_{Exponential} = \left(\sum_{i=1}^{n} P(m(T_{Jobs}) = i) \times i \right) \times \frac{1.0}{TT} \tag{4.3}$$

$$TP_{Erlang} = \left(\sum_{i=1}^{n} P(m(T_{Jobs}) = i) \times i \right) \times \frac{1.0}{TResp \times KResp} \tag{4.4}$$

O tempo médio de resposta (**RT**) pode ser obtido a partir da Lei de Little (LITTLE, 1961), como explicado na seção 3.2.1.2, desde que o sistema esteja em condição estável (JAIN, 1990) e nos conheçamos o número médio de trabalhos no sistema (N_{Jobs}) e a taxa de chegada ($ARRV_{Rate} = \frac{1}{ARRV}$). A transição da Lei de little utilizada nesse modelo é expressa na Equação 4.5. Essa expressão apenas leva em conta a taxa de chegada efetiva, desconsiderando os descartes que podem ocorrer quando não for possível aceitar nenhum trabalho.

$$M_{RT} = \frac{((\sum_{i=1}^{n} P(m(Queue) = i) \times i) + (\sum_{i=1}^{n} P(m(T_{Jobs}) = i) \times i)) \times ARRV}{1 - P((W_{Queue} = 1) \wedge (A_{Queue} = 0) \wedge (A_{Res} = 0))} \tag{4.5}$$

Também definimos uma métrica para computar o custo da infraestrutura através do uso de VMs sob demanda e reservadas. O uso de VMs elásticas (isto é, VMs sob demanda) pode ser obtido pela quantidade máxima de VMs sob demanda (**MAX_VM_OD**) menos o valor esperado de VMs sob demanda não utilizadas (no lugar $\mathbf{A_{OD}}$). Então, o valor esperado do uso de VMs elásticas (EU) é definida na Equação 4.6.

$$EU = MAX\_VM\_OD - \sum_{i=1}^{n} P(m(A_{OD}) = i) \times i \tag{4.6}$$

O custo total da infraestrutura dado em um intervalo de tempo de observação **T** (em anos), é obtido através da Equação 4.7, onde a quantidade de VMs reservadas é $\mathbf{VM_{Res}}$ e o custo de uso por um ano de VMs reservadas e sob demanda é VM_{C_Res}, VM_{C_Ond}.

$$Cost(T) = EU \times T \times VM_{C\_Ond} + VM_{Res} \times T \times VM_{C\_Res} \tag{4.7}$$

Essas equações podem ser usadas para avaliar distintos cenários de transcodificação de vídeo na nuvem, permitindo aos administradores alterarem essas variáveis e observarem o seu impacto no desempenho e no custo. Por fim, as métricas 4.3,4.4, 4.5 e 4.6 na sintaxe do Mercury podem ser vistas na Tabela 4.3.

4.2 Modelo Nuvem Privada

O modelo de transcodificação em nuvem privada é apresentada na Figura 4.2, e a descrição de seus parâmetros e os atributos de suas transições estão nas Tabela 4.4 e 4.5,

Tabela 4.3: Métricas do modelo público - Mercury

Equação	Sintaxe do Mercury
4.3	$TP_{Exponential} = E\{\#T_{Jobs}\} * \frac{1.0}{TT}$
4.4	$TP_{Erlang} = E\{\#T_{Jobs}\} * \frac{1.0}{TResp*KResp}$
4.5	$M_{RT} = \frac{(E\{\#Queue\}+E\{\#T_{Jobs}\})*ARRV}{1-P\{(\#W_{Queue}=1)AND(\#A_{Queue}=0)AND(\#A_{Res}=0)\}}$
4.6	$EU = MAX\_VM\_OD - E\{\#A_{OD}\}$

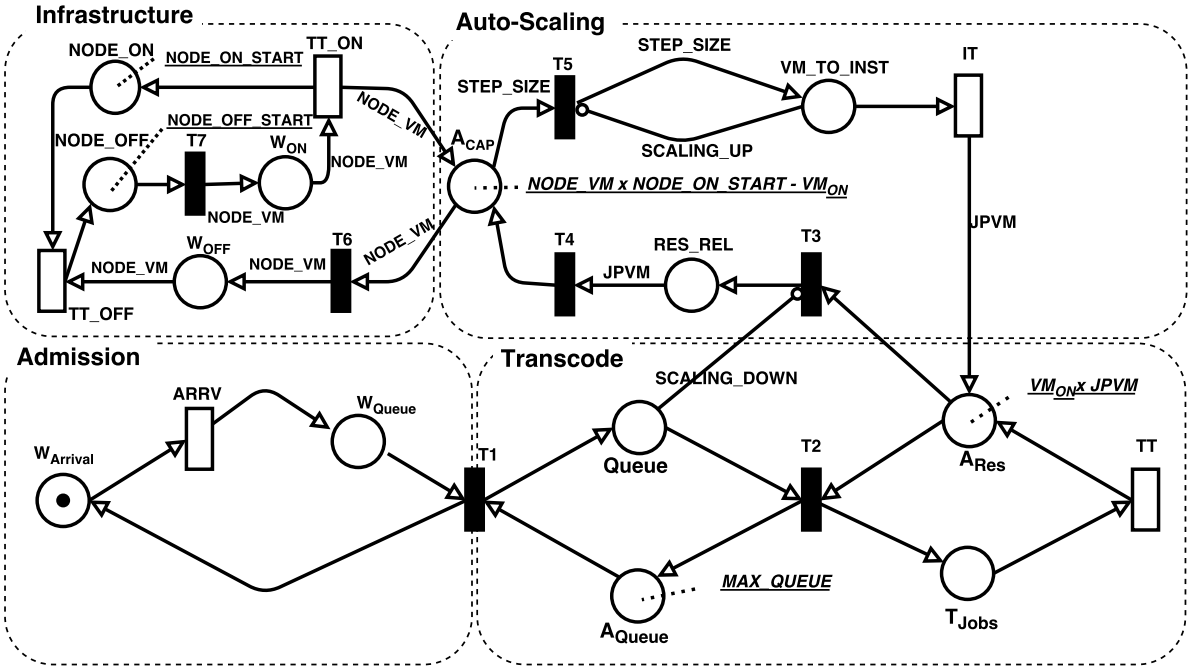


Figura 4.2: Modelo de transcodificação de vídeo em nuvem privada

respectivamente. Este modelo compartilha grande parte do fluxo e condições do modelo de nuvem pública, então apresentaremos nessa seção apenas os mecanismos e fluxos diferentes.

Uma das mais importantes distinções é que no modelo privado haverá maior percepção da limitação dos recursos, pois o número de servidores físicos que manterão as VMs são limitados, e também cada servidor será capaz de suportar uma quantidade limitada de VMs. Adotaremos neste trabalho servidores de igual capacidade, então todos os servidores serão capazes de suportar o mesmo número de VMs de um mesmo tipo. A quantidade de VMs suportada pelo servidor é definida pela variável **NODE_VM**. É importante notar que **NODE_VM** mudará para cada tipo de VM analisada, um mesmo servidor suportará um número menor de VMs que aloquem mais recursos e uma quantidade maior que aloquem em menor quantidade. Exemplo: um servidor com 4 CPUs e 8 GB de RAM disponíveis para virtualização, suportaria 2 VMs *t2.medium* ou 4 VMs *t2.small* (definidas na Tabela 3.1) . Note que esse modelo também adota tipos homogêneos de VM.

O modelo da Figura 4.2 representa um sistema capaz de se adequar e consolidar a carga

Tabela 4.4: Descrição das variáveis - modelo privado.

Variável	Descrição
$W_{Arrival}$	Espera pelo disparo de novas transcodificações.
ARRIVAL	Tempo entre chegada de trabalhos.
W_{Queue}	Espera pela disponibilidade na fila.
Queue	Trabalhos na fila.
Max_Queue	Capacidade máxima da fila do sistema.
A_{Queue}	Capacidade disponível na fila.
A_Res	Recursos de Transcodificação disponíveis.
T_Jobs	Quantidade de transcodificações sendo realizadas.
TT	Tempo de transcodificação.
JPVM	Quantidade máxima de trabalhos simultâneos por VM.
A_{CAP}	Capacidade disponível para instanciar novas VMs.
Scaling_UP	Condição para instanciar VMs.
Scaling_Donw	Condição para destruir VMs.
Step_Size	Quantidade de VMs requisitadas por *scaling up*.
VM_TO_Inst	Quantidade de VMs sendo instanciadas.
VM_{ON}	Número de VMs ligadas no início do sistema.
THR_INST	Limiar de instanciação de VMs.
THR_DEST	Limiar de destruição de VMs.
NODE_VM	Número de VMs do tipo escolhido para o servidor.
NODE_ON_START	Número de servidores ligados no início do sistema.
NODE_OFF_START	Número de servidores desligados no início do sistema.
NODE_ON	Número de servidores ligados.
NODE_OFF	Número de servidores desligados.
TT_ON	Tempo para ligar um servidor.
TT_OFF	Número de desligar um servidor.
W_{ON}	Capacidade do servidor esperando para ser adicionada ao sistema.
W_{OFF}	Capacidade do servidor em espera para ser retirada do sistema.
ThresholdShutdownNode	Limiar para desligar um servidor.
ThresholdStartNode	Limiar para ligar um servidor.
P_{Static_ON}	Potência necessária para manter um servidor ligado.
P_{Static_OFF}	Potência necessária para manter um servidor desligado.
MEC_{Power_on}	Consumo elétrico médio para ligar um servidor.
MEC_{Power_off}	Consumo elétrico médio para desligar um servidor.
$MEC_{StartVM}$	Consumo elétrico médio para ligar uma VM.

no menor número de servidores. A sub-rede de infraestrutura é responsável por definir dinamicamente a capacidade de instanciação de VMs disponível, que é dada pela quantidade de *tokens* no lugar $\mathbf{A_{Cap}}$. No início do sistema, deve ser definida a quantidade de servidores ligados através da variável **NODE_ON_START**, essa quantidade irá definir a capacidade disponível para adicionar novas VMs, representada pelo Equação: $\mathbf{NODE\_VM} \times \mathbf{NODE\_ON\_START} - \mathbf{VM_{ON}}$. Nessa capacidade inicial, foi extraída a quantidade de VMs que inicialmente deverão estar ligadas, representado pela variável $\mathbf{VM_{ON}}$, esse valor irá proporcionar a capacidade de transcodificação no início do sistema, ou seja, a

Tabela 4.5: Atributos das transições - modelo público

Transição	Tipo	Semantica	Peso	Prioridade
ARRV	Timed	Single Server	-	-
TT	Timed	Infinite Server	-	-
IT	Timed	Infinite Server	-	-
TT_ON	Timed	Infinite Server	-	-
TT_OFF	Timed	Infinite Server	-	-
T1	Immediate	-	1	1
T2	Immediate	-	1	1
T3	Immediate	-	1	2
T4	Immediate	-	1	1
T5	Immediate	-	1	1
T6	Immediate	-	1	1
T7	Immediate	-	1	1

quantidade de *tokens* no lugar $\mathbf{A_{Res}}$, ($\mathbf{VM_{ON}} \times \mathbf{JPVM}$). Durante a instanciação e destruição de VMs, a capacidade disponível em $\mathbf{A_{Cap}}$ será alterada, e a quantidade de marcas será utilizada como parâmetro para ligar ou desligar novos servidores. O momento de ligar servidores é definido pela função de habilitação em **T7** (Condição 4.8), que verificará se a capacidade de instanciação de VMs ($\mathbf{A_{Cap}}$) somada com a capacidade que está sendo adicionada (*tokens* em $\mathbf{W_{on}}$) é menor ou igual ao limiar para ligar o servidor (**ThresholdStartNode**).

$$m(A_{Cap}) + m(W_{on}) < ThresholdStartNode \tag{4.8}$$

Se a função de habilitação for verdadeira e existir pelo menos um servidor desligado (*tokens* em **NODE_OFF**), um deles receberá a requisição para entrar em operação, com a retirada de uma marca de **NODE_OFF** e o depósito de *tokens* em $\mathbf{W_{on}}$, onde ficará até que o servidor adicione sua capacidade ao sistema, ou seja, iniciado seu SO, *hypervisor*, e tenha sido reconhecido pela plataforma de gerenciamento de nuvem. O tempo para que esse processo ocorra é representado pelo tempo para disparo da transição **TT_ON**, que irá adicionar capacidade de instanciar novas VMs, inserindo **NODE_VM** *tokens* em $\mathbf{A_{Cap}}$, e também adicionando uma nova máquina em **NODE_ON**.

Por outro lado, a sub-rede de infraestrutura também irá desligar servidores quando houver excesso de capacidade não utilizada. Isso é dado pelo disparo da transição **T6**, que é habilitada quando houver excedente de capacidade, isto é, se a quantidade de *tokens* em $\mathbf{A_{Cap}}$ for maior que o limiar para desligar um servidor **ThresholdShutdownNode**, essa verificação ocorrerá pela função de habilitação da transição **T6** (Condição 4.9), que verifica também se já foi adicionado um servidor para suportar a carga. No caso de habilitação, **NODE_VM** *tokens* serão retirados da capacidade disponível e adicionados em $\mathbf{W_{OFF}}$, onde o servidor aguardará o tempo para o seu desligamento, que é representado pelo tempo para o disparo da transição **TT_OFF**. Com o disparo de **TT_OFF** o servidor se tornará um recurso disponível para ligar quando um novo

limiar para ligar for alcançado. Note que a capacidade total da infraestrutura é dada pela soma dos servidores ligados e desligados.

$$m(A_{Cap}) >= (ThresholdShutdownNode) AND (m(NODE\_OFF) < node\_off\_start) \quad (4.9)$$

Outra diferença é uma condição adicional para o *SCALING_DOWN*, que deve verificar se a quantidade de servidores ligados mudou, ou uma VM adicional foi inserida para transcodificação. Isso evitará o disparo inicial das transições imediatas, mantendo a consistência do modelo.

Nesse modelo o custo é focado no consumo elétrico, que é decorrente da potência necessária para: manter os servidores ligados e desligados; manter as VMs ligadas; o processo de ligar os servidores; o processo de desligar os servidores; o processo de ligar as VMs; e o consumo de cada VM enquanto realiza transcodificações de vídeo. Como visto na Seção 2.3, manter um servidor ligado tem um consumo elétrico associado, mesmo quando não está realizando nenhum processamento, então esse consumo será dado pela Equação 4.10, onde a potência é produto do valor esperado de *tokens* no lugar **NODE_ON** com a potência média consumida $\mathbf{P_{Static}}$ por um servidor em um intervalo de tempo T. Outro consumo associado, é manter um servidor desligado em *stand-by* para que possa ser ligado por algum método, como o *Wake-on-LAN - Wol*. Esse consumo considera o valor esperado de servidores desligados (*tokens* em **NODE_OFF**) multiplicado pela potência estática de um servidor desligado no intervalo de tempo T, como pode ser visto pela Equação 4.11.

$$ECServer_{ON}(T) = \left(\sum_{i=1}^{n} P(m(NODE\_ON) = i) \times i \right) \times P_{Static\_ON} \times T \quad (4.10)$$

$$ECServer_{OFF}(T) = \left(\sum_{i=1}^{n} P(m(NODE\_OFF) = i) \times i \right) \times P_{Static\_OFF} \times T \quad (4.11)$$

Também haverá o consumo associado a cada vez que o servidor for ligado e desligado, esse valor considera a quantidade de vezes que um servidor será ligado e desligado, para isso, utilizaremos as vazões das transições **TT_ON** e **TT_OFF**, multiplicadas pelo consumo elétrico médio necessário para ligar (MEC_{Power_ON}) e desligar (MEC_{Power_OFF}) o servidor, respectivamente. Então, teremos o consumo elétrico para ligar um servidor na Equação 4.12 e para desligar na Equação 4.13.

$$EC_{Power\_ON}(T) = \left(\sum_{i=1}^{n} P(m(W_{ON}) = i) \times i \right) \times \frac{T \times MEC_{Power\_on}}{TT\_ON} \quad (4.12)$$

$$EC_{Power\_OFF}(T) = \left(\sum_{i=1}^{n} P(m(W_{OFF}) = i) \times i \right) \times \frac{T \times MEC_{Power\_off}}{TT\_OFF} \tag{4.13}$$

De forma similar, ligar uma máquina virtual também possui um custo associado, que é dado pela vazão da transição **IT** e o consumo elétrico médio para iniciar uma VM junto com a aplicação de transcodificação, que é dado pela Equação 4.14. Não consideramos o consumo elétrico para desligar uma VM, visto que em nossos experimentos, tanto o tempo quanto o consumo foram desprezíveis.

$$EC_{StartVM}(T) = \left(\sum_{i=1}^{n} P(m(VM\_TO\_INST) = i) \times i \right) \times \frac{T \times MEC_{StartVM}}{IT} \tag{4.14}$$

Por fim, devemos considerar os consumos oriundos das transcodificações e de manter VMs ligadas sem processamento. Como pode ser visto pela Equação 2.3, a potência elétrica das VMs é decorrente da utilização dos recursos de CPU, memória e disco. Neste trabalho, utilizamos *storages* externos, então não iremos considerar o consumo vindo do disco, portanto para identificar o consumo elétrico é necessário identificar o uso de memória e CPU das VMs instanciadas, estando elas, ociosas ou não. No modelo, a capacidade de processamento das VMs pode ser representada por *tokens* em $\mathbf{A_{Res}}$ e $\mathbf{T_{Jobs}}$, sendo a primeira como a capacidade ociosa e a segunda a capacidade aplicada na transcodificação de um vídeo. Sabemos também que a utilização dos recursos de uma VM está relacionada com a quantidade de transcodificações simultâneas realizadas, como pode ser visto nas Figuras 3.5 e 3.6. Então, podemos criar duas relações, que recebam a quantidade de transcodificações realizadas por uma VM e retornem à utilização de memória e CPU, $u_{cpu}\colon \mathbb{N} \to \mathbb{R}, u_{mem}\colon \mathbb{N} \to \mathbb{R}$, então:

$$U^{CPU} = u_{cpu}(x) \tag{4.15}$$

$$U^{MEM} = u_{mem}(x) \tag{4.16}$$

Essas funções irão retornar o uso de CPU e memória para a quantidade de transcodificações simultâneas x. Por outro lado, temos que a potência de uma VM pode ser dada pela Equação 2.4, adaptando essa Equação para uma VM e os recursos utilizados no modelo, temos a Equação 4.17:

$$P_{VM}(U^{CPU}, U^{MEM}) = \alpha \times U^{CPU} + \beta \times U^{MEM} + e \tag{4.17}$$

Podemos compor as equações 4.15 e 4.16 em 4.17, e teremos a Equação 4.18 que recebe como entrada a quantidade de trabalhos simultâneos e retorna a potência necessária para transcodificação. É importante observar que uma VM ociosa normalmente tem uma mínima

utilização de processador e de memória, então $x = 0$, representa a potência de uma VM sem trabalhos.

$$P_{VM}(x) = \alpha \times u_{cpu}(x) + \beta \times u_{mem}(x) + e \tag{4.18}$$

Com isso, devemos identificar como está distribuída as probabilidades de transcodificações, consideramos que as VMs instanciadas receberão trabalhos igualmente. Isto é, para um **JPVM** maior que um, se houver 2 transcodificações a serem realizadas, cada VM receberá um trabalho. Utilizamos a Equação 4.19, que utiliza a Equação 4.18 para computar a potência da VM para a quantidade de trabalhos dada. O primeiro somatório, de $i = 1$ até n (onde n é a quantidade máxima de VMs no sistema, no modelo sendo computado na Equação 4.20) identifica probabilidade de existirem i VMs em determinado instante. O segundo somatório irá identificar as probabilidades das VMs existentes estarem realizando k transcodificações, sendo esse valor influenciado pela quantidade máxima de trabalhos por VM (**JPVM**). O último somatório calcula a potência dos k trabalhos sendo transcodificados divididos por cada VM.

$$P\_VMs = \sum_{i=1}^{n} \left(P\left(\frac{m(A_{Res}) + m(T_J) + m(R_{rel})}{JPVM} = i \right) \times \left(\sum_{k=0}^{i \times JPVM} P(m(T_{Jobs}) = k) \times \left(\sum_{t=k}^{k+i} P_{VM}\left(\left\lfloor \frac{t}{i} \right\rfloor \right) \right) \right) \right) \tag{4.19}$$

$$n = (NODE\_ON\_START + NODE\_OFF\_START) \times NODE\_VM \tag{4.20}$$

Um exemplo de parte do desenvolvimento da Equação 4.19 é apresentado nas Equações 4.21 e 4.22, quando é definido que cada VM pode realizar até dois trabalhos simultâneos (**JPVM=2**). A Equação 4.21 mostra o início do somatório, isto é, a probabilidade de existir uma VM instanciada, nesse caso essa VM pode estar realizando de 0 até **JPVM** (configurado como 2), e somado com a probabilidade de haver duas VMs (Equação 4.22). Na Equação 4.22 observamos a probabilidade de existirem duas VMs, nesse caso pode não existir nenhum trabalho, onde o custo será de duas VMs sem trabalho, para um trabalho, teremos o custo de uma VM trabalhando e outra ociosa, assim por diante. Esse processo será executado até onde for possível existirem VMs no sistema. Portanto, essa Equação dará a potência média total das VMs através do produto das probabilidades de manter VMs e seus trabalhos executados.

$$\begin{aligned} (P(\frac{m(A_{Res}) + m(T_{Jobs}) + m(R_{rel})}{2} = 1)) \times ((P(m(T_{Jobs}) = 0) \times (P_{VM}(0))) + \\ (P(m(T_{Jobs}) = 1) \times (P_{VM}(1))) + \\ (P(m(T_{Jobs}) = 2) \times (P_{VM}(2)))) + \end{aligned} \tag{4.21}$$

$$
\begin{aligned}
(P(\frac{m(A_{Res})+m(T_{Jobs})+m(R_{rel})}{2}=2))\times((P(m(T_{Jobs})=0)\times((P_{VM}(0))+(P_{VM}(0))))+\\
(P(m(T_{Jobs})=1)\times((P_{VM}(0))+(P_{VM}(1))))+\\
(P(m(T_{Jobs})=2)\times((P_{VM}(1))+(P_{VM}(1))))+\\
(P(m(T_{Jobs})=3)\times((P_{VM}(1))+(P_{VM}(2))))+\\
(P(m(T_{Jobs})=4)\times((P_{VM}(2))+(P_{VM}(2)))))+
\end{aligned}
\tag{4.22}
$$

O consumo elétrico das VMs será dado pelo produto da Equação 4.19 pelo tempo **T**, como pode ser visto na Equação 4.23. O consumo elétrico total do sistema será dado pela soma das equações 4.10, 4.11, 4.12, 4.13, 4.14 e 4.23, como mostra a Equação 4.24. Podemos também concluir que a chave para economia é identificar adequadamente os parâmetros que reduzam a quantidade de servidores ligados, a quantidade de VMs ociosas e também redução da quantidade de vezes que são ligados e desligados os servidores e VMs, além de identificar os valores que minimizem o custo considerando os compromissos entre: manter servidores desligados e o consumo de ligar e desligar para atender a variação da demanda; juntamente com manter VMs ligadas em relação ao custo de liga-las. Onde também devem ser considerados os SLAs de desempenho.

$$
ECVMs(T) = P\_VMs \times T \tag{4.23}
$$

$$
\begin{aligned}
TotalPowerConsumption(T) =& ECServer_{ON}(T)+ECServer_{OFF}(T)+\\
& EC_{Power\_ON}(T)+EC_{Power\_OFF}(T)+\\
& EC_{StartVM}(T)+ECVMs(T)
\end{aligned}
\tag{4.24}
$$

As métricas de desempenho desse modelo são as mesmas utilizadas no modelo de nuvem pública. As métricas utilizadas para o cálculo do consumo elétrico na sintaxe da ferramenta Mercury (SILVA et al., 2015), podem ser observadas na Tabela 4.6, com exceção da Equação 4.19, que pode ser gerada utilizando o algoritmo no Anexo B.

4.3 Validação do Modelo Nuvem Pública

Esta subseção descreve os experimentos computacionais utilizados para verificar a precisão do modelo de nuvem pública. Ele visa garantir que o modelo represente o sistema real e seja capaz de prover resultados corretos com 95% de confiança. A avaliação foi realizada em um ambiente de nuvem privada que emprega a API Amazon EC2 (AWS, 2016) e também com máquinas virtuais com configuração equivalente às configurações da Amazon, isto permite que o

Tabela 4.6: Métricas do modelo privado - Mercury

Equação	Sintaxe do Mercury
4.10	$ECServer_{ON}(T) = E(\#NODE\_ON) * P_{Static\_ON} * T$
4.11	$ECServer_{ON}(T) = E(\#NODE\_OFF) * P_{Static\_OFF} * T$
4.12	$EC_{Power\_ON}(T) = E(\#W_{ON}) * \frac{T * MEC_{Power\_on}}{TT\_ON}$
4.13	$EC_{Power\_ON}(T) = E(\#W_{OFF}) * \frac{T * MEC_{Power\_off}}{TT\_OFF}$
4.14	$EC_{StartVM}(T) = E(\#VM\_TO\_INST) * \frac{TMEC_{StartVM}}{IT}$
4.19	Anexo B

modelo validado seja estendido a nuvens públicas.

4.3.1 Arquitetura de Teste

A infraestrutura física para validação possui dois nós de computação com CPU Xeon E3-1220V3: 4 x 3.10 GHz, 8 MB Cache,Memória RAM de 32 GB, 2 *NIC* GigabitEthernet. Foi instalado o *hypervisor* XenServer 6.5 (XENSERVER, 2016) nos servidores, que são usados para suportar as VMs, O ambiente também possui dois *Storages* com quatro HDs em RAID 5, que mantém as imagens e discos das VMs, como também os vídeos transcodificados.

Nós utilizamos um *Switch* com 16 portas GigabitEthernet e capacidade máxima de 32Gbps. Nós utilizamos o gerenciador de nuvem Cloudstack (CLOUDSTACK, 2016) em um servidor com Core i7 CPU, 4 x 3.40 GHz, 8 MB Cache, 4 GB RAM, 1 GigabitEthernet NIC. O Front-End foi instalado em uma VM do tipo *t2.small*, com a configuração indicada na Tabela 3.1. Para requisitar as transcodificações foi utilizado um computador cliente com processador i7 , 4 x 3.40 GHz, 8 MB Cache, 4 GB RAM, NIC 1 GigabitEthernet. O *software* gerador de requisições utilizado foi o Jmeter (JMETER, 2016). A arquitetura pode ser vista na Figura 4.3.

O sistema de transcodificação de vídeo utilizado para os testes foi desenvolvido em Java, sendo composto por dois subsistemas que se comunicam por meio de *Sockets* para manter o comportamento do diagrama de sequência da Figura 3.4. O primeiro subsistema é o *Front-end* que recebe as requisições dos clientes, realiza o balanceamento de carga e o *auto-scaling*. As chamadas do *auto-scaling* são realizadas através da API EC2 para o gerente da nuvem.

O sistema de transcodificação recebe as requisições do *Front-end*, realiza as transcodificações e informa o estado atual para o *Front-end*. Esse subsistema usa FFMPEG (FFmpeg, 2016) para transcodificar os vídeos recebidos nos mais diversos formatos para o container **MP4** com codec *h264*, um dos formatos recomendados para disponibilização na Web (W3C, 2010; W3SCHOOLS, 2016). Os vídeos transcodificados são armazenadas no *storage*.

Dois cenários foram utilizados para validar o modelo. No Cenário I, utilizamos diversos vídeos do mesmo tamanho, com duração de 00:04:55, frames do tamanho (640x356), 25 frames

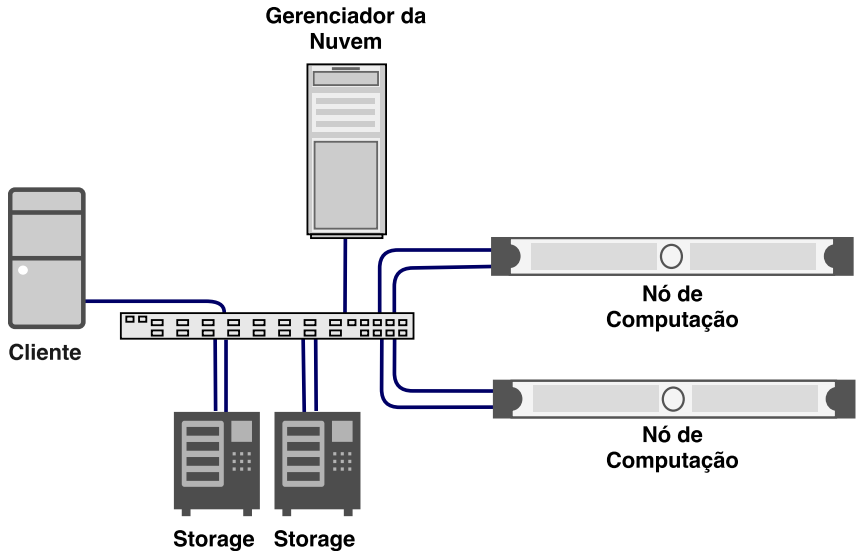


Figura 4.3: Infraestrutura de nuvem pública

sendo exibidos por segundo e taxa de 200kbps. O Cenário II utiliza vídeos de diferentes durações, sendo a média das durações iguais às médias encontradas no YouTube (CHENG; LIU; DALE, 2013), com frames do tamanho (640x360), 48 por segundo e taxa de 448 kbps.

Para ambos os cenários, nós definimos os mesmos parâmetros tanto no sistema quanto no modelo SPN de nuvem pública (Figura 4.1). O tipo de VM usado na transcodificação foi *t2.small*; a quantidade máxima de VMs elásticas foi 3, apenas uma VM reservada, o valor para o limiar de instanciação foi 4, o limiar para destruição das VMs sob-demanda foi 2, o *step-size* e o número de conversões simultâneas por VM (**JPVM**) foi 1.

4.3.2 Resultados Experimentais e Validação

A configuração mencionada é suficiente para executar o sistema real, mas o modelo necessita de dois outros importantes valores: o tempo de transcodificação quando **JPVM = 1** para a VM **t2.small**; e o tempo para instanciação da VM, incluindo a distribuição desses parâmetros. A fim de identificar a duração e a distribuição do tempo para realizar transcodificações, utilizamos o Jmeter para gerar carga no nosso sistema de transcodificação de vídeo na nuvem. Nós configuramos o Jmeter para enviar 60 trabalhos de conversão intercalados por 1 minuto de pausa depois do fim de cada transcodificação, então, nós obtivemos o valor do tempo de transcodificação **TT** sem que haja influência do enfileiramento do servidor. No caso de vídeos com diferentes tamanhos, eles foram escolhidos aleatoriamente durante a execução do experimento. A Tabela 4.7 apresenta a informação da distribuição de probabilidade em ambos os cenários. Em ambos os cenários pelos métodos Anderson-Darling e Kolmogorov-Smirnov com 95% de confiança (ANDERSON; DARLING, 1954; CHAKRAVARTI; LAHA, 1967), não há evidência para refutar que a distribuição para o tempo de transcodificação seja Erlang.

O próximo parâmetro a ser medido no ambiente de teste na nuvem foi o tempo de instanciação da VM. Esse tempo depende do tipo da VM. A Tabela 4.8 apresenta os resultados

Tabela 4.7: Tempo de transcodificação da infraestrutura da Figura 4.3

Cenário	Média (s)	Distribuição	Fases	Taxa
Cenário I	22,34	Erlang	350	0,0638
Cenário II	15,58	Erlang	9	1,6843

dos testes para a VM **t2.small**. Esses valores foram medidos enviando 60 requisições EC2 para a criação de novas instâncias **t2.small** e observando o tempo que essa VM levou para se tornar apta para realizar transcodificações. As requisições foram realizadas pelo algoritmo no Anexo D. Os métodos Anderson-Darling e Kolmogorov-Smirnov também não indicaram qualquer evidência para refutar que o tempo de instanciação siga a distribuição Erlang, com 95% de confiança.

Tabela 4.8: Tempo para instanciação da infraestrutura da Figura 4.3

Cenário	Média (s)	Distribuição	Fases	Taxa
Cenário I and II	21,14	Erlang	522	0.040

Depois de obter os dados de entrada necessários para o modelo, nos executamos a validação usando três diferentes tempos entre chegada de requisições, tanto para o modelo quanto para o sistema. No Cenário I, foram empregados tempos entre chegadas exponencialmente distribuídos com 35, 20 e 10 segundos (Embora o Jmeter não ofereça a funcionalidade dessa distribuição, nós ajustamos esse comportamento através do script constante no Anexo C). O Jmeter executou 30 vezes para cada tempo entre chegada, em cada execução 100 vídeos foram convertidos. Portanto, os resultados representam o comportamento do sistema considerando um total de 9.000 transcodificações. A vazão e o uso de VMs elásticas foram medidos durante a validação dos experimentos.

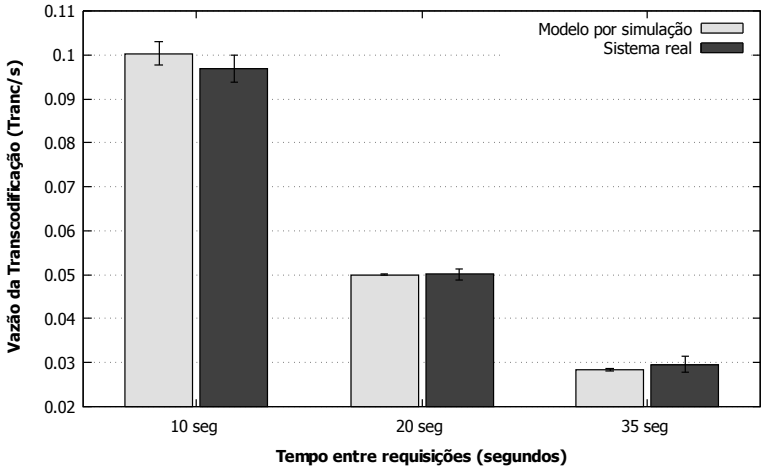


Figura 4.4: Cenário I - Validação da Vazão - Distribuição Erlang

As transições temporizadas possuem muitas fases, para evitar a explosão do espaço de estados, o modelo inicialmente foi executado por meio de simulação estacionária, considerando as distribuições Erlang mencionadas para tempos de transcodificações e instanciação. A Figura

4.4 mostra a comparação da resultado da vazão medida no sistema real e o modelo resolvido por simulação. Os resultados obtidos através do modelo foram consistentes com os resultados do sistema real, para os três casos de tempo entre chegada, pois, além dos intervalos de confiança com 95% se sobreporem, o teste t para os três casos incluíram o zero: para 10 segundos (-0,000360305; 0,006964413); para 20 segundos (-0,000929275; 0,000971685); e para 35 segundos (-0,002883069; 0,000238826).

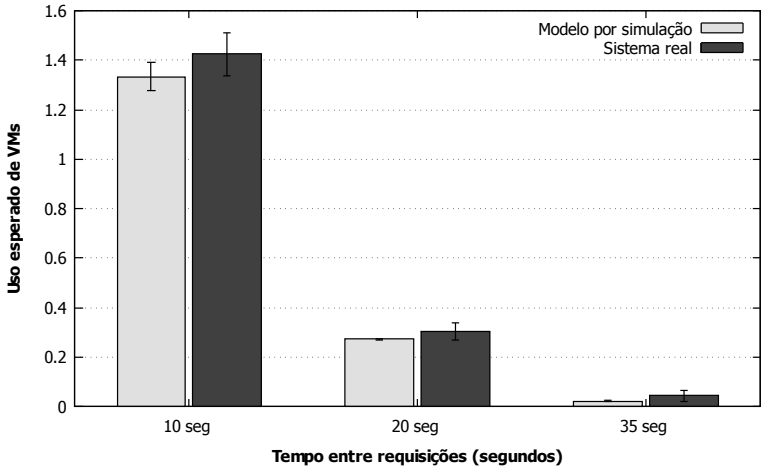


Figura 4.5: Cenário I - Validação do uso de VM elástica - Distribuição Erlang

Já a Figura 4.5 retrata a comparação para o uso esperado de VMs elásticas. Para essa métrica, o modelo proposto também possui resultados equivalentes ao comportamento do sistema real. Os três tempos entre requisições mostram que os intervalos de confiança também se sobrepõem juntamente com a inclusão do zero no teste t para os três casos: em 10 segundos (-0,182912567;0,00072387); em 20 segundos (-0,069081232; 0,002675001); e em 35 segundos (-0,042033207; 0,001513236). Então, nós podemos justificadamente confiar no modelo considerando a precisão do intervalo de confiança de 95%.

Tabela 4.9: Cenário I - Validação por análise numérica

Chegada	Medida	I.C. do Sistema	Média do sistema	Análise
10 s	Vazão (transc/s)	(0,0938; 0,1000)	0,096960	0,09999
	Uso de VMs elásticas	(1,3386; 1,5099)	1,424200	1,3765
20 s	Vazão (transc/s)	(0,0487; 0,0514)	0,050061	0,0500
	Uso de VMs elásticas	(0,2686; 0,3401)	0,304400	0,3384
35 s	Vazão (transc/s)	(0,0277; 0,0314)	0,029631	0,0285
	Uso de VMs elásticas	(0.0186; 0,0664)	0,042500	0,0607

Em uma análise complementar, nós verificamos a adoção de transições exponenciais no modelo, visando realizar análise numérica em vez de simulação para calcular as métricas desejadas, sem que ocorra a explosão do espaço de estados devido à quantidade de fases das Erlangs (consideração que será especialmente útil durante as várias execuções necessárias para o processo de otimização). Embora os tempos para transcodificação e instanciação não sejam

exponencialmente distribuídos, a Tabela 4.9 mostra que o modelo ainda nos dá resultados confiáveis usando distribuições exponenciais nas transições. Os valores para a vazão (em transcodificações por segundo) e uso esperado de VMs elásticas (em número de VMs) estão dentro do intervalo de confiança do sistema, o que não nos permite refutar que o modelo resolvido por analise numérica também representa o sistema com 95% de confiança.

A validação do Cenário II foi conduzida com a mudança dos tempos entre chegadas para 10, 15 e 20 segundos. Adicionalmente, os vídeos foram selecionados aleatoriamente durante a execução do Jmeter. A vazão pode ser vista na Figura 4.6 e o uso de VMs elásticas na Figura 4.7. Este segundo cenário também executou um total de 9.000 conversões. Como os valores da simulação do modelo estão dentro do intervalo de confiança das respectivas medidas do sistema juntamente com a inclusão do zero no teste t para a vazão nos tempos entre chegadas: de 10 segundos (-0,000341995; 0,008467368); de 15 segundos (-0,000438642; 0,004007637); e de 20 segundos (-0,002623737; 0,00135145). O mesmo também pode ser observado para o uso esperado de VM para os tempos entre chegadas de: 10 segundos (-0,177595784; 0,003988524); 15 segundos (-0,065354421; 0,003465229); e 20 segundos (-0,073608393; 0,000930706). Nós também podemos dizer que o modelo representa o sistema com 95% de confiança para esse caso.

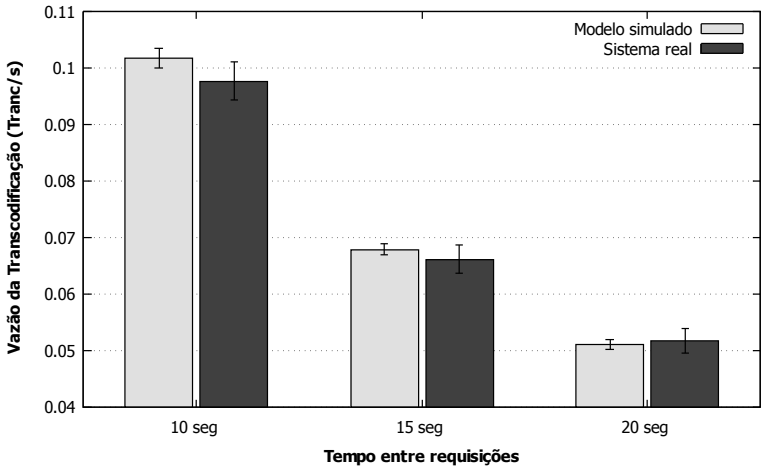


Figura 4.6: Cenário II - Validação da vazão - Distribuição Erlang

Também foram comparados os experimentos no Cenário II para análise numérica (isto é, utilizando distribuições exponenciais). A Tabela 4.10 mostra que os intervalos de confiança para ambas as métricas, vazão e uso de VMs elásticas, contêm o valor computado via análise numérica. Portanto, mesmo sem usar distribuições Erlang para o tempo de transcodificação e instanciação, o modelo continua consistente com o comportamento para o Cenário II.

4.4 Validação do Modelo Nuvem Privada

Esta seção tem o objetivo de demonstrar uma validação adicional, verificando que as métricas de consumo elétrico do modelo de nuvem privada também apresentam resultados

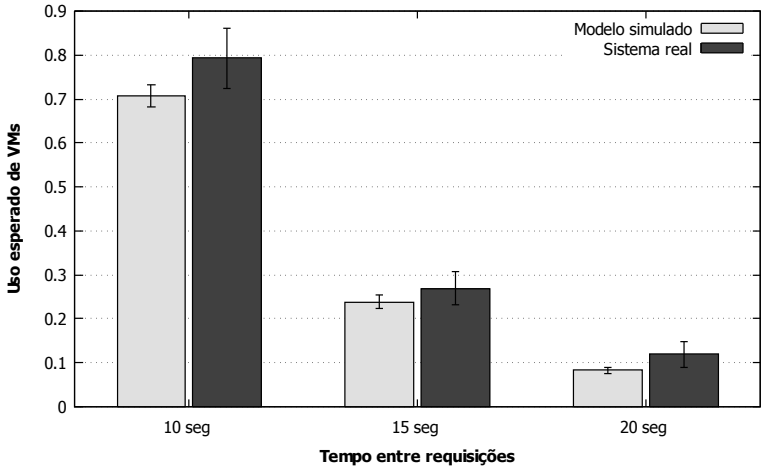


Figura 4.7: Cenário II - Validação do uso de VM elástica - Distribuição Erlang

Tabela 4.10: Cenário II - Validação por análise numérica

Chegada	Medida	I.C do sistema	Média do sistema	Análise
10 s	Vazão (transc/s)	(0,0943; 0,1010)	0,09769	0,0999
	Uso de VMs elásticas	(0,7252; 0,8618)	0,7935	0,7688
15 s	Vazão (transc/s)	(0,0636; 0,686)	0,06616	0,0666
	Uso de VMs elásticas	(0,2318; 0,3079)	0,2694	0,3059
20 s	Vazão (transc/s)	(0,0495; 0,0538)	0,05172	0,0499
	Uso de VMs elásticas	(0,0899; 0,1479)	0,1189	0,1324

confiáveis quando comparado com sistemas reais. O consumo elétrico foi avaliado no modelo e comparado com o intervalo de confiança de 95% obtido por medições no sistema, também foi verificado se o tempo necessário para transcodificação de 30 vídeos no modelo estava dentro do intervalo de confiança do tempo para o sistema.

4.4.1 Arquitetura de Teste

A infraestrutura é composta por 2 nós de computação com processador Core i7 CPU, 4 x 3.40 GHz, 8 MB Cache, 8 GB RAM, 1 GigabitEthernet NIC, nesses nós foram instalados o *hypervisor* XenServer 6.5, e são os responsáveis por suportar as VMs utilizadas. O ambiente também possui um *storage* com quatro HDs em RAID 5 e 2 GigabitEthernet NIC, que mantém as imagens e discos das VMs, como também os vídeos transcodificados.

O gerenciador de nuvem CloudStack foi instalado em um computador com Core i5 CPU, 4 x 2,66 GHz, 8 MB Cache, 8 GB RAM, 1 GigabitEthernet NIC. Usamos também um computador cliente com Core i5 CPU, 4 x 3,20 GHz, 4 MB Cache, 4 GB RAM, 1 GigabitEthernet NIC para enviar as requisições de transcodificações pelo Jmeter. A interligação de rede é realizada por um switch *Switch* privado com 16 portas GigabitEthernet e capacidade máxima de 32Gbps. Para medição elétrica foi utilizado o PDU *Watts UP*, que intermedeia e monitora a entrega de energia elétrica para os dois nós de computação, como mostra a Figura 4.8.

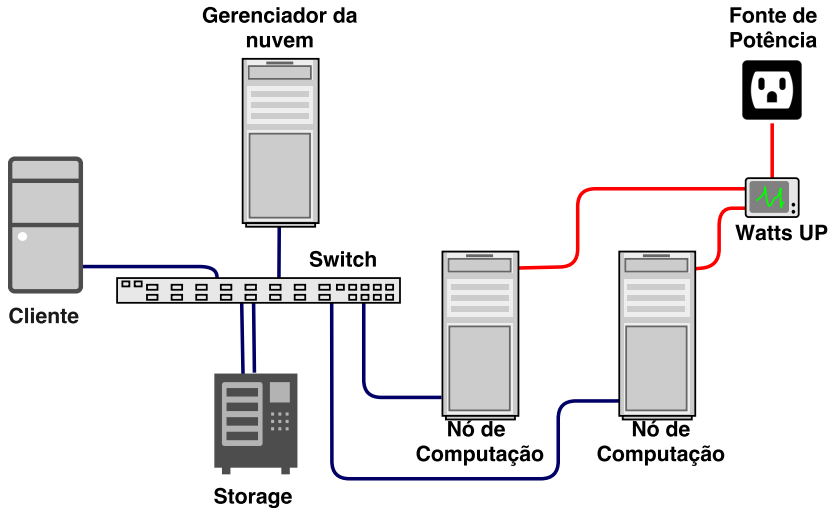


Figura 4.8: Infraestrutura de nuvem privada

Essa validação utiliza um cenário com os mesmos vídeos do Cenário II da validação em nuvem pública, isto é, o cliente enviará diversos vídeos de diferentes durações no container AVI, sendo a média das durações iguais às médias encontradas no YouTube, com frames do tamanho (640x360), 48 por segundo e taxa de 448 kbps, esses vídeos são transcodificados em vídeos no container .MP4 com codec h264. O processo de envio, balanceamento de carga e controle do *auto-scaling* foi realizado pela mesma aplicação com dois subsistemas da seção anterior.

A configuração que foi inserida no modelo e no sistema de transcodificação de vídeo real é apresentada na Tabela 4.11. É importante observar que para o tipo de VM *t2.small* cada servidor pode oferecer até 3VMs, utilizando 3 dos 4 núcleos dos processadores disponíveis, e 6 dos 8 GB de RAM, os outros 2 GB disponíveis serão utilizados para manter o SO e o *hypervisor*. Também é importante ressaltar que adotamos tempos entre chegada de 45 segundos exponencialmente distribuídos.

Tabela 4.11: Configuração de validação para nuvem privada

Parâmetro	**Valor**
ARRIVAL	40 segundos
Tipo de VM	t2.small
JPVM	1
A_{CAP}	5
Step_Size	1
VM_{ON}	1
THR_INST	5
THR_DEST	2
NODE_VM	3
NODE_ON_START	2
NODE_OFF_START	0
ThresholdShutdownNode	0
ThresholdStartNode	10

4.4.2 Resultados Experimentais e Validação

Da mesma forma da validação na nuvem pública, a configuração da Tabela 4.11 permite a execução do sistema, entretanto, o modelo necessita de configurações adicionais para as transições temporizadas e o modelo elétrico. No Modelo 4.2, existem quatro transições temporizadas, **TT**, **IT**, **TT_ON**, e **TT_OFF**. Os tempos das transições**TT** e **IT** dependem do tipo de VM e infraestrutura utilizada. O tempo para transcodificação foi medido através do Jmeter com 60 requisições para transcodificações intervaladas de um minuto para que a medição não sofra o efeito de filas, o valor médio e a distribuição encontrada podem ser vistos na Tabela 4.12. Já o tempo para instanciação de VMs foi medido utilizando o script do Anexo D e o valor da distribuição e os parâmetros podem ser vistos na Tabela 4.12.

Tabela 4.12: Tempo de transcodificação e instanciação - Nuvem privada

Parâmetro	Média (s)	Distribuição	Fases	Taxa
TT	20,3958	Erlang	15	1,325
IT	27,18	Erlang	1292	0,02103

O **TT_ON**, e **TT_OFF** dependem do *hardware* físico dos servidores. O valor de **TT_ON** leva em consideração o momento que o servidor é ligado até o instante que o gerenciador da nuvem aceita o servidor para sua infraestrutura. Foi medido considerando o tempo obtido pelo comando *uptime* do Linux, que recupera a informação do tempo que o computador foi ligado. O Tempo que o servidor foi reconhecido pelo gerenciador da nuvem foi obtido pelo log de Cloudstack. Com a diferença dos dois tempos foi possível obter o **TT_ON**, esse processo foi repetido 30 vezes e a distribuição e média podem ser vistas na Tabela 4.13.

Tabela 4.13: Tempo para ligar um servidor - Nuvem privada

Parâmetro	Média (s)	Distribuição	Taxa
TT_ON	100,38	Exponencial	0,0041

O **TT_OFF** foi obtido em conjunto com o medidor elétrico *Watts UP*. Foi requisitado o desligamento do servidor por *Secure Shell - SSH*, registrado esse tempo, e depois comparado com o tempo que o servidor deixou de consumir potência elétrica. A diferença desses dois valores permitiu identificar o tempo para desligar o servidor. Esse procedimento foi executado 30 vezes, o tempo e a distribuição podem ser vistos na Tabela 4.14.

Tabela 4.14: Tempo para desligar um servidor - Nuvem privada

Parâmetro	Média (s)	Distribuição	Fases	Taxa
TT_OFF	50,30	Erlang	172	0,29088

Além dos tempos, o modelo privado requer uma série de medições relativas ao consumo elétrico de estados do sistema, os valores dessas medições serão utilizados para obter os

parâmetros $\mathbf{P_{Static\_on}}$, $\mathbf{P_{Static\_off}}$, $\mathbf{MEC_{Power\_on}}$, $\mathbf{MEC_{Power\_off}}$ e $\mathbf{MEC_{StartVM}}$ para utilização nas Equações 4.10, 4.11, 4.12, 4.13 e 4.14, respectivamente.

Para obtenção do valor de $\mathbf{P_{Static\_on}}$, foi medida a potência instantânea utilizada por um servidor ligado no PDU como mostra o Anexo A, onde foram colhidas 400 amostras. O valor de $\mathbf{P_{Static\_off}}$, foi medido com o mesmo servidor desligado e conectado no PDU. Também foram colhidas 400 amostras. Os valores para a potência de um servidor ligado e desligado podem ser vistos na Tabela 4.15.

Tabela 4.15: Potência média medida do servidor da infraestrutura da Figura 4.8 sem carga

Parâmetro	Valor (W)
$\mathbf{P_{Static\_on}}$	37,7
$\mathbf{P_{Static\_off}}$	3,78

Os valores de: consumo elétrico médio para ligar o servidor ($\mathbf{MEC_{Power\_on}}$); consumo elétrico para desligar um servidor $\mathbf{MEC_{Power\_off}}$; e o consumo para ligar uma VM $\mathbf{MEC_{StartVM}}$ foram medidos juntamente com a medição dos tempos para essas atividades através do *Watts UP*. Foram colhidas 30 amostras para cada. Os valores para essas atividades estão na Tabela 4.16.

Tabela 4.16: Custo elétrico médio medido por evento na infraestrutura da Figura 4.8

Parâmetro	Valo (Wh)
$\mathbf{MEC_{Power\_on}}$	1,21
$\mathbf{MEC_{Power\_off}}$	0,567
$\mathbf{MEC_{StartVM}}$ - t2.small	0,151

É também necessário obter os parâmetros α β e e da Equação (4.19). Para isso devemos obter valores da potência elétrica instantânea em função da utilização de CPU e memória da VM *t2.small*. Com esse objetivo, utilizamos um script em *bash Linux* , que usa o *nmap* para monitorar o *hardware* enquanto o *stress* (STRESS, 2016) gerava uma carga variável de CPU e a memória. Também utilizamos o *cpulimit* para ajustar o uso percentual de CPU (CPULIMIT, 2016).

O sistema foi monitorado com o *Watts UP* e os valores da potência subtraídos da potência de manter o servidor ligado $\mathbf{MEC_{Power\_on}}$, e depois ordenados pelo tempo com os dos usos e memória e CPU para serem utilizados em uma regressão linear. A Equação obtida pode ser vista na Equação 4.25, onde os valores de α, β e e são 20.012, 0.649 e 1.241, respectivamente, essa regressão obteve o R^2 ajustado de 96,50%, o que nos diz que o nosso modelo com memória e CPU explica quase toda a variação da potência.

$$P_{VM}(x) = 20.012 \times u_{cpu}(x) + 0.649 \times u_{mem}(x) + 1.241 \tag{4.25}$$

A Equação 4.25 possui como entrada a quantidade de trabalhos na VM, então nós precisamos relacionar o uso de memória e CPU com a quantidade de trabalhos realizados por

VM. Para isso, deve ser obtida a relação da utilização de CPU e memória necessárias para transcodificar até **JPVM** vídeos simultâneos (utilizado na Equação 4.18) para a VM, no nosso caso, a utilização de memória e processador da VM do tipo **t2.small** para transcodificar 0 e 1 vídeos, esses valores foram obtidos com o *software nmon* durante as 60 requisições para obtenção do **TT** e podem ser vistos na Tabela 4.17.

Tabela 4.17: Uso de recurso por JPVM da VM **t2.small** - Nuvem privada

JPVM	$\mathbf{u_{cpu}(x)}\%$	$\mathbf{u_{mem}(x)}\%$
0	0	18,09
1	76,6	80,8

De posse de todos os valores das transições e equações necessárias para a computação das métricas podemos comparar o resultado do modelo e do sistema. Foi desenvolvido um teste que consiste na medição do tempo e da potência. A medição do tempo leva em conta a duração da transcodificação de 30 vídeos com tempos entre chegada de 40 segundos exponenciais (usando o script do Anexo C) através do Jmeter. A potência média é medida com o *Watts UP* para esse período para os dois servidores usados. Esse teste é executado 30 vezes, portanto foram executadas 900 transcodificações.

Já o modelo, foi executado por simulação com a distribuições definidas da tabelas acima. É importante notar que computamos o uso da VM Front-end, que é necessária para o sistema, essa VM não teve processamento significativo (0,11%) e manteve também uso estável de memória em 50%, adicionando 1,5875132 Watts ao valor encontrado no modelo SPN.

Os intervalos de confiança com 95% de significância, para o tempo de transcodificação de 30 vídeos e a potência média do modelo e do sistema podem ser vistos na Figura 4.9 e 4.10, respectivamente. É importante notar que o intervalo de confiança da simulação é muito pequeno para aparecer na imagem. Os dados da simulação e do sistema também foram comparados pelo teste t não pareado e o zero estava dentro do intervalo obtido tanto para a potência média (-0,907492799; 0,214878163) quanto para o tempo para codificar 30 vídeos (-77,47927503; 33,08449103). Portanto, não temos podemos refutar o modelo representa o sistema com 95% de confiança.

Também identificamos o comportamento do modelo utilizando análise numérica, embora as transições **TT**, **IT** e **TT_OFF** não sejam exponenciais, iremos considerar exponenciais no modelo, isso permite evitar a explosão do espaço de estados proveniente das técnicas de *moment matching*.

A comparação da análise numérica do modelo com o sistema pode ser vista na Tabela 4.18, tanto o tempo necessário para transcodificação de 30 vídeos, como a potência média do modelo estão dentro do intervalo de confiança. O que não permite refutar, com 95% de confiança, que as utilizações de distribuições exponenciais nas transições do modelo também nos dão uma aproximação adequada do comportamento do sistema.

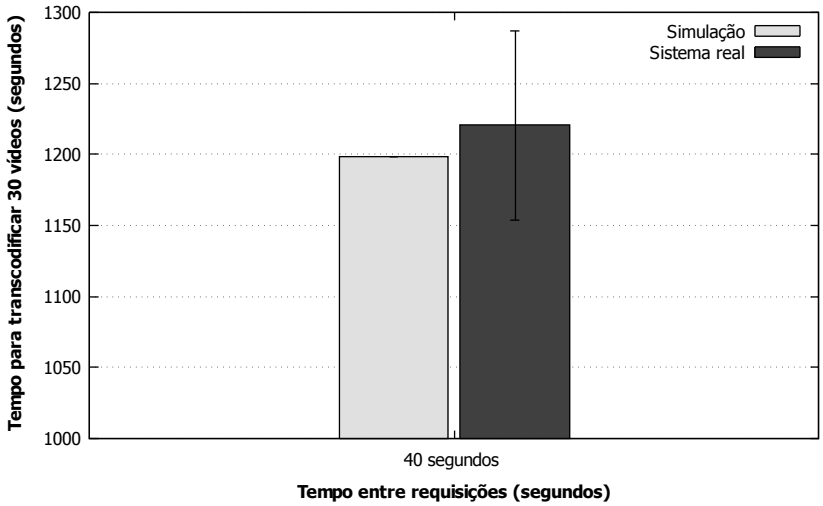


Figura 4.9: Validação - Tempo para converter 30 vídeos

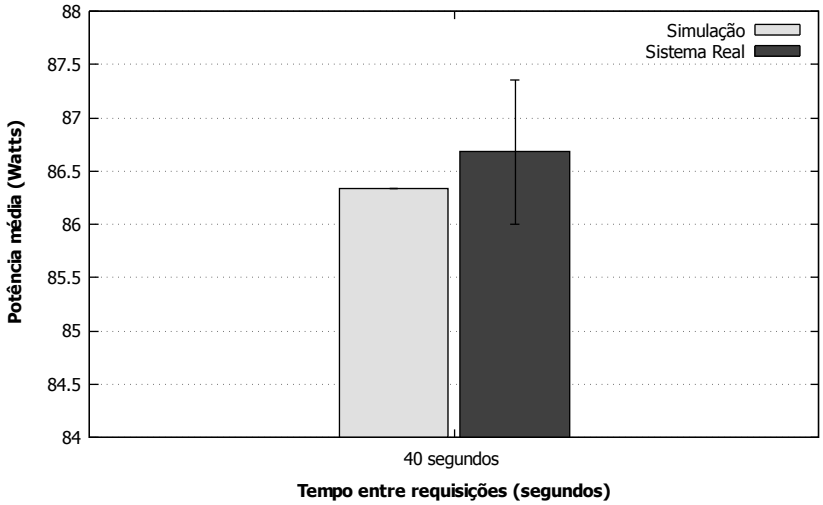


Figura 4.10: Validação - Potência média

4.5 Otimização em Modelos

Os modelos possuem diversos parâmetros e possibilidades de combinações de valores dos parâmetros, essas combinações apresentam valores diferentes para as métricas do modelo, explorar o espaço de soluções através da variação individual de cada valor dos parâmetros em busca de uma configuração otimizada é uma tarefa exaustiva, em muitos casos impraticável. Devido a isso utilizamos mecanismos de otimização. Esta seção explora o espaço de soluções possíveis para o sistemas de nuvem pública e privada, buscando identificar os valores que devem ser configurados em cada parâmetro para se alcançar uma configuração que respeite o SLA e otimize o custo. Esse processo utiliza o algoritmo de otimização GRASP, que foi adaptado para buscas pelas variáveis do modelo e adotando como função objetiva as métricas do modelo.

O algoritmo de otimização utilizado receberá como entrada: o modelo; os parâmetros que deverão ser configurados; as possibilidades de variação de cada parâmetro; as restrições das

Tabela 4.18: Validação por análise numérica - infraestrutura privada

Chegada	medida	I.C do sistema	Média do sistema	Análise
40 s	Tempo para transcodificar (s)	(1154,1; 1287,3)	1220,7	1200
	Potência (W)	(86,003; 87,355)	86,679	86,1475132

métricas de desempenho (vazão e tempo médio de resposta mínimo); a carga de trabalho que o sistema é submetido; e os valores associados à nuvem escolhida. Depois disso o algoritmo irá gerar soluções a partir do conjunto de possibilidades que otimizem o custo.

Utilizaremos o Algoritmo 1 para otimização, ele possui duas fases, a construção e a busca local. A fase de construção deve gerar uma boa solução semi-gulosa, que aplicada ao modelo, irá selecionar para cada parâmetro um valor dentro de um intervalo finito de elementos que comporá uma configuração válida para o que minimize o custo e respeite as restrições.

4.5.1 Construção de soluções para o modelo

Nossa aplicação da fase de construção no modelo, pode ser vista no Algoritmo 5. Ela recebe como entrada: o modelo da Figura 4.1 para nuvem pública ou o modelo da Figura 4.2 para nuvem privada; os parâmetros com os possíveis valores internos (exemplo: VM_R, que poderá variar de 1 até 30; ou tipos de VM que podem ser t2.micro, t2.large, cada uma com seus tempos de transcodificação e instanciação para a carga de trabalho esperada; e etc.); α que irá determinar o tamanho da lista restrita de candidatos (RCL); β que será utilizada para aumentar a variabilidade de soluções (COLMENAR et al., 2016); a carga de trabalho esperada para o sistema; e o SLA de tempo médio de resposta e vazão.

Algoritmo 5: Construction Model

Input: model, parameters[], α, β, SLA

repeat
  $standardSolution \leftarrow generateRandomCandidate(parameters[])$;
  $selectedConfiguration[] \leftarrow \emptyset$;
  $i \leftarrow 0$;
  while $(i < parameters.size())$ **do**
    $i \leftarrow i+1$;
    $parameterValues[] \leftarrow chooseRandomValues(parameters[], i, \beta)$;
    $evaluededSortedElements[] \leftarrow$
     $evaluateElements(model, parameterValues, standardSolution)$;
    $selectedElement \leftarrow randomRCL(evaluatedSortedElements[], \alpha)$;
    $selectedConfiguration[i] \leftarrow selectedElement$;
  end
  $constructionSolution \leftarrow stationarySolve(model, selectedConfiguration[])$;
until $(isvalid(constructionSolution, SLA))$;
return $constructionSolution$

Result: $constructionSolution$

A linha 2 gera uma solução padrão a partir da escolha aleatória de um valor de todos os possíveis para cada parâmetro. A partir da linha 5 até a linha 11, serão selecionados os parâmetros pelo método semi-guloso, a variável i definirá qual parâmetro será escolhido na iteração. A linha 7 selecionará aleatoriamente parte dos valores possíveis para o parâmetro i, se β for 1, todos os possíveis valores serão escolhidos, e se for 0.5, metade dos valores, com o mínimo de 1. Essa estratégia serve para aumentar a variabilidade das soluções geradas na fase de construção (COLMENAR et al., 2016). *chooseRandomValues* também deverá selecionar apenas os valores viáveis para o parâmetro, no caso da nuvem pública o valor do limiar de destruição não deve ser maior ou igual ao limiar de instanciação.

Após obter os valores do parâmetro i a serem testados, a linha 8 irá substituir cada um deles na solução padrão e avaliar o modelo, obtendo a variação do custo decorrente de cada um dos valores, para então ordenar os parâmetros do menor para o maior custo incremental e retornar uma lista com os parâmetros ordenados pela variação do custo. É importante ressaltar que a avaliação do modelo é realizada através da chamada da API do Mercury (SILVA et al., 2015) e a solução dada pelo solução estacionária pelo método de GAUSS.

A função *ramdomRCL* na linha 9, recebe a lista com os valores e custos ordenados do menor para o maior, e então, seleciona um valor aleatoriamente dentro da lista restrita de candidatos (RCL). A RCL é composta pelos melhores da lista $evaluatedSortedElements[]$, sendo seu tamanho dado por α, 1 para todos os elementos e 0 para apenas 1. É importante ressaltar que essa é a principal componente aleatória do GRASP para evitar os mínimos locais. O elemento selecionado da RCL será usado como o valor para o parâmetro i da solução (linha 10). Esse processo será repetido para todos os parâmetros até que seja composta a solução, que será inserida no modelo e avaliado numericamente pela API do Mercury, onde serão geradas as métricas de vazão, tempo médio de resposta e custo. A validade dessa configuração é avaliada na linha 13, que serve para evitar que sejam enviadas para busca local soluções que não respeitem o SLA mínimo esperado.

4.5.2 Soluções Locais Para o Modelo

A busca local tentará melhorar a solução encontrada na fase de construção, reduzindo o custo da solução. Nós utilizamos três diferentes abordagens de solução local, simples, *Variable Neighborhood Descent - VND* (onde a vizinhança da busca local muda a cada melhoria), e uma versão customizada do VND que serão apresentadas a seguir.

4.5.2.1 Busca Local Simples

O Algoritmo 6 apresenta a busca local simples. Ele recebe o modelo; a solução válida gerada na fase de construção; a definição do tamanho da vizinhança γ; e a quantidade máxima de iterações. Ele inicia definindo que a melhor solução, é a recebida na construção (linha 1), e será executado até que tenha realizado $maxIteration$ iterações (linha 3). A função *getNeighbor*

da linha 5, irá buscar uma configuração na vizinhança da solução encontrada na construção. A vizinhança para os parâmetros do modelo considera uma parcela do intervalo total de valores para cada parâmetro, onde γ irá definir a quantidade de elementos. A utilização de γ igual a 0.2, de um parâmetro com 10 elementos significa que a busca local irá variar até dois valores do valor encontrado na fase de construção.

Algoritmo 6: Local Search Model

Input: model, constructionSolution,γ, maxIteration

```
1  minSolution ← constructionSolution;
2  i ← 0;
3  while (i < maxIteration) do
4      i ← i + 1;
5      neighborConfiguration[] ← getNeighbor(constructionSolution, γ);
6      neighborSolution ← stationarySolve(model, neighborConfiguration[]);
7      if (isValid(neighborSolution) AND neighborSolution < minSolution) then
8          minSolution ← neighborSolution;
9          return minSolution;
10     end
11 end
12 return minSolution
13 ; Result: minSolution
```

Nessa busca local, todos os parâmetros poderão ser mudados, para então, realizar a análise estacionária do modelo com os novos parâmetros. Se a solução for válida e possuir um menor custo que a anterior, então, ela passará a ser a nova solução (linha 8). O algoritmo termina quando encontrar uma melhoria ou quando o número máximo de iterações for executado. É importante notar que a busca local também não irá buscar elementos em toda vizinhança, pois a quantidade de possibilidades de variação dentro de uma mesma vizinhança também será muito grande e o método de primeira melhoria para esse tipo de busca normalmente apresenta os mesmos resultados que a busca em toda a vizinhança, só que em menor tempo (GENDREAU; POTVIN, 2010).

4.5.2.2 Busca Local VND

A busca local VND utilizará a busca local, e mudará o centro da busca a cada iteração. O objetivo dessa proposta é melhorar a qualidade das soluções ao mesmo instante que muda um ponto com menores custos locais. O Algoritmo 7 apresenta essa estratégia aplicada ao modelo. Ele recebe como entrada: o modelo; a solução da fase de construção; γ, que definirá o intervalo da variação; e a quantidade máxima de buscas dentro da busca local.

As linhas 1 e 2 iniciam as variáveis que serão utilizadas no modelo da mesma forma

que a busca local simples. Porém, diferente da busca local simples a quantidade máxima de execução desse algoritmo não será determinístico pela quantidade máxima de iterações, e sim pela quantidade máxima de não melhorias, que será dado pela variável *maxNonImprovement*, pois a variável de controle de iteração *i* será reiniciada a cada melhoria e também reinício de uma nova vizinhança.

A linha 4 irá realizar uma simples busca local a partir da vizinhança de *minSolution*, caso encontre uma nova solução viável que minimize a solução (condição da linha 5), a nova solução será a mínima, portanto, a vizinhança de busca local será a da nova solução, também será reiniciada a variável de iteração (linha7). Caso a solução não melhore, uma nova iteração será feita e uma nova busca local realizada. Note que a variabilidade das soluções que não encontram melhoria, é devido ao aspecto aleatório existente na função *localSearch*. Outro aspecto importante são as sucessivas mudanças de vizinhanças nas condições de melhoria.

Algoritmo 7: Local VND Search Model

Input: model, constructionSolution,γ, maxNonImprovment,maxLocalSearch

```
1  minSolution ← constructionSolution;
2  i ← 0;
3  while (i < maxNonImprovement) do
4  |   newSolution ← simpleLocalSearch(model, minSolution, γ, maxLocalSearch);
5  |   if (isValid(newSolution) AND newSolution < minSolution) then
6  |   |   minSolution ← newSolution;
7  |   |   i ← 0;
8  |   else
9  |   |   i ← i + 1;
10 |   end
11 end
12 return minSolution
```

Result: *minSolution*

4.5.2.3 Busca Local VND Customizada

A última estratégia de busca local implementada, visa apoiar-se nas tendências de melhoria dos parâmetros quando são aumentados ou diminuídos para os modelos estudados. Então, criamos um algoritmo que como o VND, muda de vizinhança a cada melhoria. Porém, a busca por novas melhorias utilizará a memória da última vizinhança para reduzir o número de possibilidades.

A estratégia de busca local é apresentado no Algoritmo 8, que recebe as mesmas entradas que a busca local simples. Também inicia as linhas 1 e 2 da mesma maneira. Porém, a linha 3 apresenta a variável *directions*, que inicia vazia. A quantidade de iterações depende da quantidade de não melhorias encontradas nas vizinhanças (linha 4), o que pode denotar um mínimo local.

Algoritmo 8: Customized Local VND Search Model

Input: model, constructionSolution,γ, maxIteration,maxNonImprovment

1 $minSolution \leftarrow constructionSolution$;
2 $i \leftarrow 0$;
3 $directions \leftarrow \emptyset$;
4 **while** $(i < maxNonImprovement)$ **do**
5 $\quad neighborConfiguration[] \leftarrow getNeighbor(minSolution, directions, \gamma)$;
6 $\quad neighborSolution \leftarrow stationarySolve(model, neighborConfiguration[])$;
7 $\quad$ **if** $(isValid(neighborSolution)$ *AND* $neighborSolution < minSolution)$ **then**
8 $\quad\quad directions \leftarrow generateDirections(neighborSolution, minSolution)$;
9 $\quad\quad minSolution \leftarrow neighborSolution$;
10 $\quad\quad i \leftarrow 0$;
11 $\quad$ **else**
12 $\quad\quad directions \leftarrow \emptyset$;
13 $\quad\quad i \leftarrow i + 1$;
14 $\quad$ **end**
15 **end**
16 **return** *minSolution*

Result: *minSolution*

Diferente da função *getNeighbor* do Algoritmo 6, essa função recebe o parâmetro *directions*, para a primeira iteração ela é vazia, e a função se comporta da mesma maneira que a função do Algoritmo 6, no entanto, em outras iterações *directions* poderá ser preenchida com os valores, -1, 0 e 1 para cada parâmetro do modelo. Esses valores irão definir se a variação do parâmetro na vizinhança poderá: aumentar, quando for atribuído 1; diminuir, quando for atribuído -1; e aumentar ou diminuir quando for atribuído 0.

Esses valores serão atribuídos à *directions*, quando for encontrada alguma melhoria válida (linha 7), e então, comparados cada parâmetro da solução nova e antiga pela função *generateDirections* (linha 8). Portanto, essa estratégia busca uma solução no sentido que foi realizada a última melhoria. É importante notar que não necessariamente o parâmetro irá na direção anterior, pois essa escolha é aleatória, ele poderá manter o valor anterior, ou adicionar até o máximo da variação da vizinhança (dado por γ).

Além de gerar as direções, caso ocorra uma melhoria, o novo valor mínimo será a melhor solução e as iterações serão reiniciadas. Caso não ocorra melhoria, serão descartadas as direções geradas pela última melhoria e aumentará uma iteração. O algoritmo termina quando o máximo de não melhorias for atingido, retornando a melhor solução encontrada.

4.6 Considerações Finais

Esse capítulo apresentou as principais contribuições deste trabalho, os modelos de nuvem pública e privada, bem como os pseudocódigos de otimização aplicados nos modelos. Descrevemos inicialmente o modelo e as métricas para o custo, vazão e tempo médio de resposta de uma arquitetura de nuvem pública. Em seguida, apresentamos o modelo de nuvem privada, que além do modelo SPN, utiliza o modelo energético para obtenção do consumo elétrico.

Por fim, apresentamos os pseudocódigos das fases de construção e busca local dos mecanismos de otimização aplicados nos modelos estocásticos. Na fase de busca local, exibimos três diferentes algoritmos, uma versão de busca simples, um VND, e uma versão customizada do VND.

5

Estudos de Caso

Este capítulo descreve os estudos de caso que demonstram a aplicação dos modelos propostos no Capítulo 4. O primeiro estudo de caso irá apresentar uma carga de trabalho e uma configuração inicial do sistema implantado em uma nuvem pública. A partir da configuração inicial buscaremos identificar a influência dos parâmetros no cumprimento dos SLAs e no custo. Em todos os estudos de caso utilizamos análise numérica, considerando todos os tempos entre disparo de transições exponencialmente distribuídos.

O segundo caso de uso apresenta uma variação do modelo de nuvem pública (Figura 4.1) com objetivo de identificar soluções estatísticas para o tempo de conclusão de conjuntos de trabalho, onde serão apresentadas as probabilidades que um conjunto de vídeos recebidos em uma taxa sejam transcodificados até um determinado tempo. Esse caso de uso tem o objetivo de demonstrar a flexibilidade do uso dos modelos propostos para SLAs mais restritos.

O terceiro e quarto estudos de caso apresentam as maiores contribuições desse trabalho, que é a possibilidade de encontrar quais os valores de parâmetros e contratos devem ser selecionados em nuvens públicas e privadas, para que se obtenha um sistema com custo otimizado para uma carga de trabalho esperada, e que respeitem os SLAs de desempenho (vazão e o tempo de resposta). Estes estudos de caso também apresentam a relação dos custos de diversos SLAs, oferecendo ao administrador um método confiável para negociar o custo de cada SLA.

O quinto caso de uso apresenta o estudo com as diferentes variações dos algoritmos de busca local com o GRASP, onde foram identificados os tempos necessários para encontrar uma configuração otimizada em infraestruturas de nuvem pública em cada algoritmo.

5.1 Estudo de caso I - Comportamento do custo e tempo médio de resposta

Este estudo de caso explora o comportamento do custo e do tempo médio de resposta. Ele considera a variação individual dos diversos parâmetros do sistema enquanto é submetido aos vídeos de diferentes durações utilizados na validação do Cenário II da nuvem pública

(Seção 4.3.2), com tempos entre requisições de 6,5 segundos. Além disso, nós procuramos manualmente soluções que minimizassem o custo respeitando o SLA de tempo médio de resposta de 40 segundos. Avaliamos o modelo por análise numérica considerando a configuração inicial apresentada na Tabela 5.1. Essa configuração foi sucessivamente ajustada de acordo com a melhor solução encontrada para cada parâmetro.

Tabela 5.1: Configuração inicial do Estudo de Caso I

Parâmetro	**valor**
Tipo de VM	t2.micro
Instâncias reservadas	1
Limiar de instanciação	4
Limiar de destruição	1
JPVM	1
step_size	1

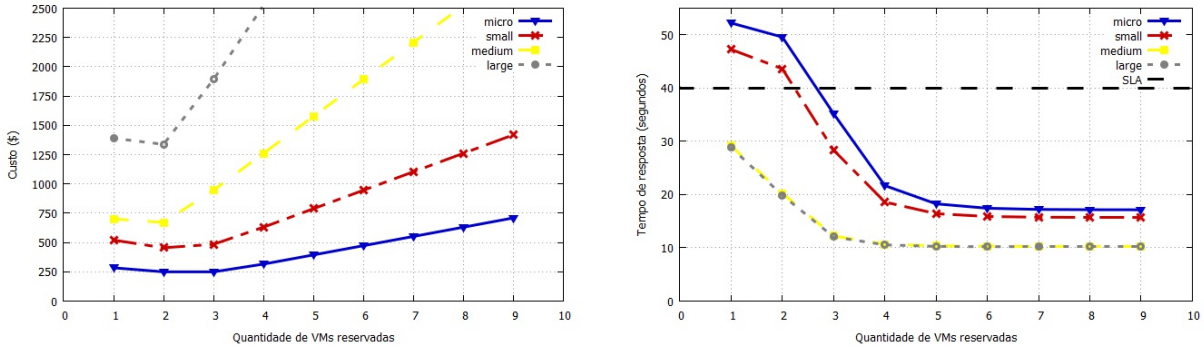
Nesta análise, foi necessário identificar os tempos para transcodificação de cada tipo de VM para 1 trabalho, e o tempo de instanciação dos diferentes tipos de VM. Os tempos de transcodificação foram obtidos com 60 medições intervaladas de um minuto e podem ser vistas na Tabela 5.2. Já o tempo de instanciação para os diferentes tipos de VM foi obtido através do *script* de requisição de VMs por EC2 do Anexo D, e os valores também podem ser vistos na Tabela 5.2.

Tabela 5.2: Tempo das transições temporizadas no Estudo de Caso I

Tipo de VM	**Tempo de Transcodificação**	**Tempo de Instanciação**
t2.micro	17,1363 segundos	21,76 segundos
t2.small	15,7066 segundos	21,14 segundos
t2.medium	10,2907 segundos	20,48 segundos
t2.large	10,1071 segundos	20,36 segundos

A Figura 5.1 (a) retrata o relacionamento entre o número de VMs reservadas e o custo para os quatro tipos de VM da Tabela 3.1. Essa análise considera a variação de 1 até 9 VMs reservadas. A VM do tipo *t2.micro* apresenta o menor custo em toda a série quando é configurado com 2 VMs reservadas, além de possuir o menor custo com o incremento de novas VMs reservadas. Por outro lado os maiores custos são das VMs *t2.medium* e *t2.large*, chegando a $2838.20 e $5676.41, respectivamente. Ambos com 9 VMs reservadas, esses valores não foram retratados no gráfico para ressaltar que o custo de possuir uma VM reservada em todos os casos é maior que o custo de possuir 2, isso se deve à utilização de VMs sob demanda, que a partir de certo uso, torna-se mais caro que manter mais VMs reservadas.

Embora possua os menores custos, *t2.micro* também possui os piores tempos de resposta, como apresentado pela Figura 5.1 (b). Tanto *t2.micro* quanto *t2.small* apenas cumprem o SLA quando pelo menos 3 VMs reservadas são usadas, já *t2.medium* e *t2.large* respeitam o SLA com qualquer número de VMs reservadas. A Figura 5.1 (a) também expõe a importância da



(a) Resultado do custo

(b) Resultado do tempo de resposta

Figura 5.1: Resultado da variação da quantidade de VMs reservadas

escolha adequada do tipo de VM, pois usar 3 VMs *t2.micro* representa apenas 18.6% do custo da configuração *t2.large* mais barata (com 2 VMs reservadas) para o cumprimento do SLA.

Nós também analisamos a configuração de diferentes quantidades de trabalhos simultâneos (**JPVM**) para cada tipo de VM. Para isso, utilizamos os tempos de transcodificação para cada tipo de VM das funções das retas da Figura 5.2. As funções foram usadas para evitar a medição de cada quantidade de transcodificação por tipo de VM. Elas foram obtidas pela regressão das quantidades de transcodificações simultâneas: 1, 3, 5, 7, 9 e 11, medidas no sistema. Esses pontos são originários de 60 requisições intervaladas de 1 minuto para transcodificações simultâneas para cada um dos tipos de VM no sistema real, apresentado na Figura 4.3. A regressão linear possui o coeficiente de determinação com mais de 99%, os valores pontuais de 1 até 11 (**JPVM**) podem ser vistos na Tabela 5.3.

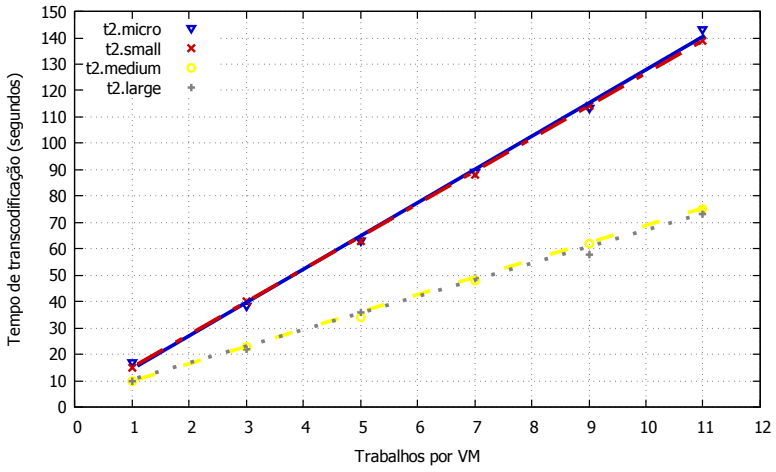


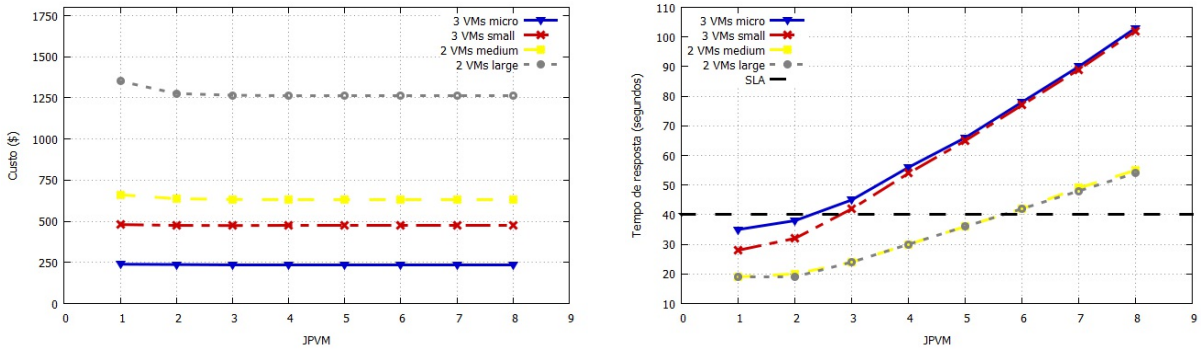
Figura 5.2: Tempo de transcodificação da Arquitetura 4.3 em relação ao **JPVM**

Então, utilizamos os diferentes tempos de transcodificação de cada tipo de VM para quantidade de trabalhos simultâneos. Também foram usados os tempos de instanciação de cada tipo de VM. Nesse teste, nós consideramos as opções mais baratas em termos de quantidade de VM reservadas que cumpriram o SLA, isto é, 3 VMs reservadas do tipo *t2.micro* e *t2.small*, e 2 dos tipos *t2.medium* e *t2.large*. Transcodificações simultâneas não apresentaram variação

Tabela 5.3: Tempo para transcodificação (JPVM) da Arquitetura 4.3

JPVM	t2.micro (s)	t2.small (s)	t2.medium (s)	t2.large (s)
1	14,44275	15,09564	9,853591	10,4859
2	27,04812	27,48588	16,41389	16,77952
3	39,65349	39,87613	22,97419	23,07314
4	52,25886	52,26638	29,53449	29,36675
5	64,86423	64,65662	36,09478	35,66037
6	77,4696	77,04687	42,65508	41,95399
7	90,07497	89,43712	49,21538	48,24761
8	102,6803	101,8274	55,77568	54,54123
9	115,2857	114,2176	62,33597	60,83485
10	127,8911	126,6079	68,89627	67,12846
11	140,4965	138,9981	75,45657	73,42208

significativa nos custos, como pode ser visto pela Figura 5.3 (a). Entretanto, houve um aumento relevante no tempo médio de resposta (Figura 5.3 (b)), as VMs *t2.micro* e *t2.small* deixaram de cumprir o SLA a partir de 3 trabalhos simultâneos, os demais tipos a partir de 6. Portanto, optamos por utilizar apenas 1 trabalho simultâneo na vm *t2.micro*.



(a) Resultado do custo

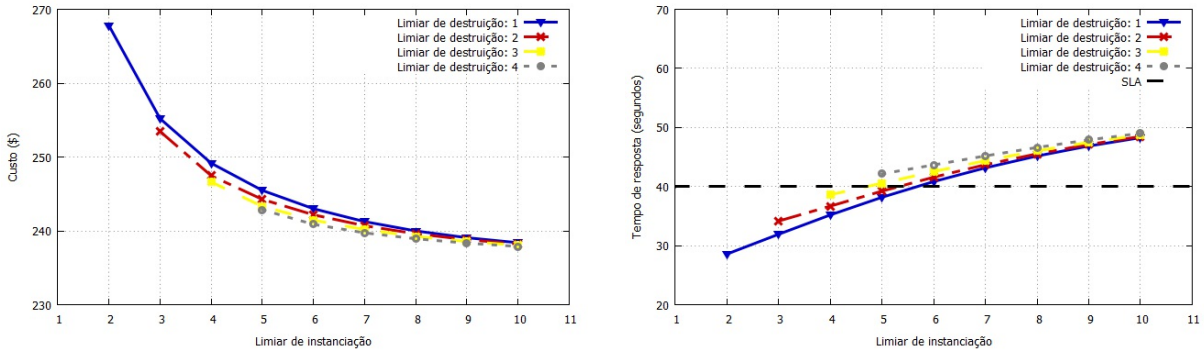
(b) Resultado do tempo de resposta

Figura 5.3: Resultado da variação da quantidade de trabalhos simultâneos

Por fim, nós verificamos se os custos podem ser reduzidos com o ajuste dos limiares de instanciação e destruição. Utilizamos como base a configuração que respeita o SLA de menor custo, ou seja, 3 VMs reservadas do tipo *t2.micro*. Nós variamos os limiares de destruição de 1 até 4, enquanto o de instanciação varia de 2 até 10, é importante notar que o limiar de instanciação deve sempre ser superior ao de destruição, isso permite que as VMs sob demandas criadas possam ser destruídas quando o pico de requisições diminuir.

A Figura 5.4 (b) apresenta a variação do desempenho em relação aos limiares. Identificamos que com o limiar de destruição 1 e instanciação 2 e 3 reduz o tempo de resposta de 35.19 para 28.57 e 31.92 segundos, respectivamente. Há também uma redução menor do tempo de resposta para o limiar de destruição 2 e o de instanciação 3 para 34.16 segundos. O custo dos limiares definidos podem ser vistos na Figura 5.4 (a). As configurações que reduziram o tempo de resposta do valor inicial também aumentaram o custo de 3% até 7%. Portanto, para essa carga

de trabalho, a melhor configuração inicial encontrada é composta de 3 VMs reservadas do tipo t2.micro, e o restante da configuração dos parâmetros da tabela inicial.



(a) Resultado do custo

(b) Resultado do tempo de resposta

Figura 5.4: Resultado da variação dos limiares de instanciação e destruição.

Podemos observar o crescimento não linear das métricas em relação a variação dos parâmetros. Também pode-se ver que a configuração adequada irá depender da carga de trabalho e SLAs do sistema.

5.2 Estudo de caso II - Distribuição do Tempo de Resposta de Transcodificação

A fim de mostrar a flexibilidade do modelo, nós apresentamos nesse estudo de caso a distribuição de probabilidade do tempo de resposta para um conjunto de trabalhos de transcodificações. Foi analisada a probabilidade de completar um conjunto de 100 trabalhos a um dado tempo. Este método pode ser aplicado para verificar a probabilidade do tempo de resposta do sistema.

O administrador do sistema pode utilizar essa abordagem para ter uma visão ampla da quantidade de trabalho que uma determinada configuração é capaz de realizar dentro de um intervalo de tempo. Esta análise requer mudanças no modelo da Figura 4.1, ele deve conter um lugar absorvente para trabalhos completados na sub rede de transcodificação (**F_J**). Também devem ser realizadas mudanças na sub rede de admissão, que não possuirá o arco de **T1** para **W_A**, e o número de *tokens* em **W_A** será *BATCH*, resultando no modelo da Figura 5.5.

Então, nós avaliamos a probabilidade de absorção nos vários pontos de tempo através de análise transiente, isto é, a probabilidade de todos os trabalhos, representados pelos 100 *tokens*, terem sido completamente processados, isto é, estarem no lugar **F_J**. Esta métrica é descrita pela Expressão **P{F_J=100}**. Isto também pode ser computado pela probabilidade da Cadeia de Markov gerada pelo modelo encontrar um estado absorvente, onde não há outro estado a ser encontrado.

A Tabela 5.4 apresenta as configurações para os três cenários comparados nesse estudo de caso. A Tabela 5.5 apresenta as métricas computadas por análise numérica para os respectivos

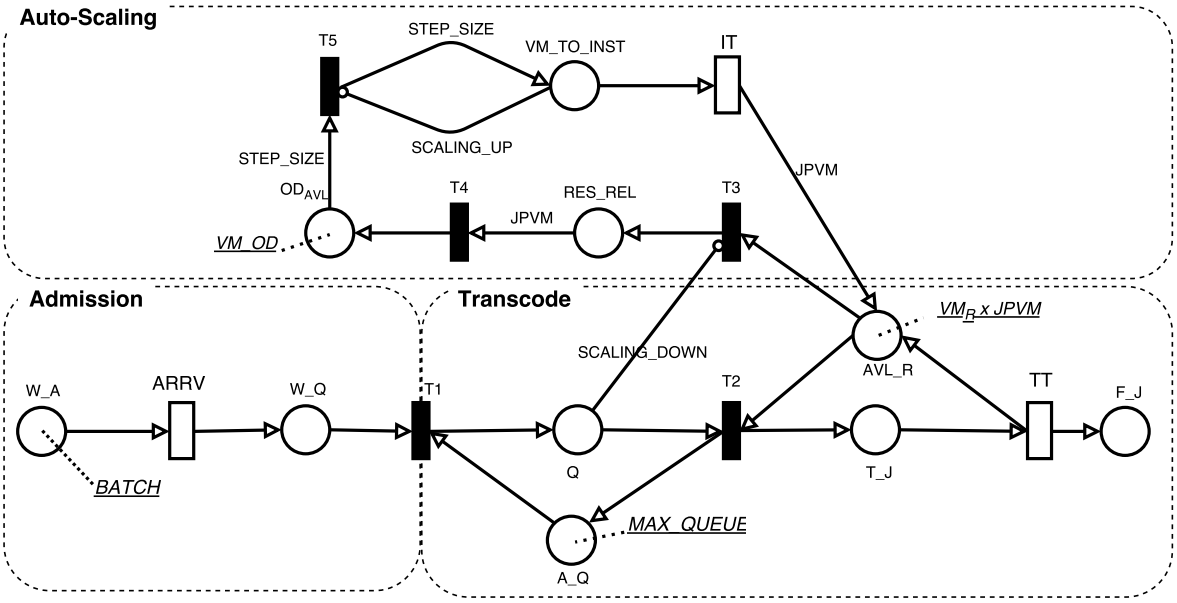


Figura 5.5: Modelo de transcodificação de vídeo com estado absorvente.

cenários utilizando o modelo da Figura 4.1. O Cenário I e II emprega 3 VMs reservadas do tipo *t2.micro* com dois diferentes limiares de instanciação, e possuindo custos de $ 255.196 e $ 267.8, respectivamente. O Cenário III usou uma configuração com 2 VMs reservadas do tipo *t2.medium*.

Tabela 5.4: Parâmetros de configuração dos cenários do Estudo de Caso II

Parâmetro	**Cenário I**	**Cenário II**	**Cenário III**
Tipo de VM	t2.micro	t2.micro	t2.medium
Instâncias reservadas	3	3	2
Limiar de instanciação	3	2	4
Limiar de destruição	1	1	1
JPVM	1	1	1
step_size	1	1	1

Tabela 5.5: Métricas dos cenários do Estudo de Caso II

Métrica	**Cenário I**	**Cenário II**	**Cenário III**
Tempo médio de resposta (s)	31,9265	28,5761	20,248
Custo ($)	255,196	267,8	671,257

A Figura 5.6 apresenta a probabilidade de completar 100 trabalhos para os três cenários. Como esperado, os Cenários mais baratos levaram mais tempo para alcançar a mesma probabilidade para completar todas as atividades. Para 80% de probabilidade de finalização: o Cenário I levou 750 segundos; o Cenário II, 741 segundos; e o Cenário III, 730 segundos. Então, a diferença de 20 segundos entre o Cenário I e o III, leva também a um aumento de cerca de 162% no custo. Este tipo de análise pode incorporar mais informações estatísticas na definição

do SLA e do tempo médio de resposta. Portanto, um provedor de serviço pode negociar os limites superiores e inferiores para grandes clientes que precisam processar alta carga de trabalho com níveis de serviço restritivos.

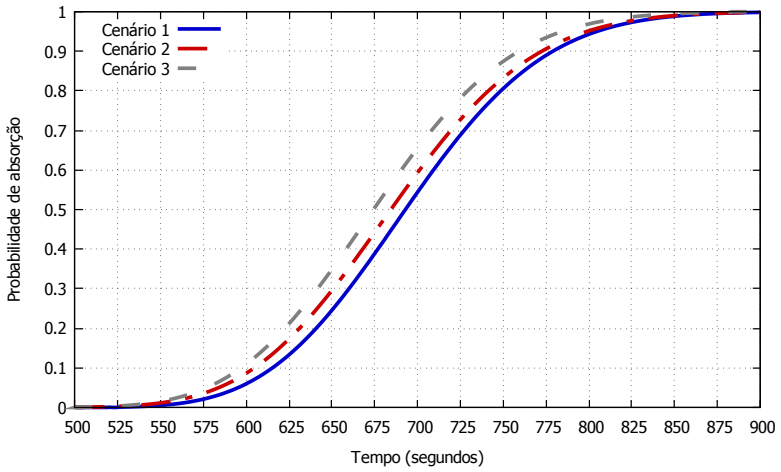


Figura 5.6: Probabilidade de absorção em relação ao tempo

5.3 Estudo de caso III - Nuvem Pública com Otimização

Este estudo de caso demostra a aplicação do modelo de nuvem pública juntamente com os algoritmos de otimização da Seção 4.5. O algoritmo de otimização irá explorar o espaço de configuração de sete parâmetros (dois dependentes do tipo de VM). Portanto o GRASP identificará: o tipo de VM a ser usado; a quantidade de VMs reservadas a serem contratadas; a quantidade de trabalhos simultâneos por VM (**JPVM**); as configurações do auto-scaling (limiar de instanciação e destruição, e o *step-size*) que devem ser configurados para minimizar o custo enquanto também cumpre o SLA definido pelo tempo médio de resposta e vazão.

Nós consideramos um cenário onde a carga de trabalho esperada é composta pelos vídeos de diferentes durações descritos na Seção 4.3.2, com tempo entre requisições de 10 segundos (exponencialmente distribuídos). O processo de otimização deverá identificar a configuração que deverá ser utilizada na nuvem para minimizar o custo do sistema em três diferentes restrições temporais (Tabela 5.6).

Tabela 5.6: Restrições (SLA) do Estudo de Caso III

Caso	Medida	Valor
1	Vazão mínima	0,099 transc/seg
	Tempo máximo de resposta	15 segundos
2	Vazão mínima	0,099 transc/seg
	Tempo máximo de resposta	30 segundos
3	Vazão mínima	0,099 transc/seg
	Tempo máximo de resposta	45 segundos

O algoritmo de otimização nos dará os valores mais apropriados para a configuração do sistema a partir da definição de um conjunto finito de possibilidades a ser analisado para cada parâmetro. Para os parâmetros: **step_size**, limiar de instanciação, limiar de destruição, **JPVM**, e o número de instâncias reservadas, nós iremos investigar a variação entre 1 e 10. Consideraremos também os tipos de VMs da Tabela 3.1.

Como os vídeos são os do Cenário II da validação, e os tipos de VM da Tabela 3.1, os tempos de transcodificação para trabalhos simultâneos dos tipos de VMs usados, são os da Figura 5.2 e Tabela 5.3. Também utilizamos o tempo de instanciação da Tabela 5.2. Todos os tempos entre os disparos das transições foram considerados exponenciais para possibilitar a análise numérica do modelo.

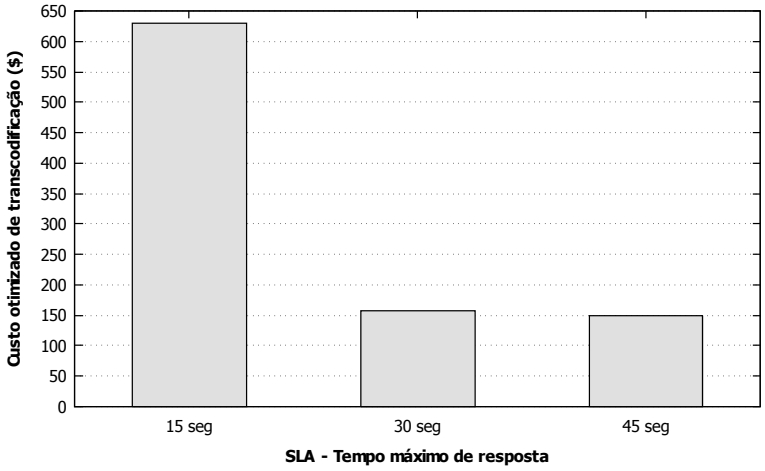


Figura 5.7: Custo dos cenários de nuvem pública otimizados

Os custos tomaram como base um ano de utilização, e os custos individuais das VMs e contratos são os da Tabela 3.1. O resultado do menor custo encontrado pelo algoritmo de otimização para cada SLA pode ser visto na Figura 5.7. Essa figura demonstra, como esperado, que quanto menor o tempo médio de resposta do SLA, maior será o custo. Entretanto, a variação entre os SLAs e os custos não são lineares, como fica evidenciado pela diferença do crescimento do custo para dois SLAs distintos. Do SLA de 15 para 30 segundos o custo cresceu mais de 299%. Já de 45 para 30 segundos, o custo apenas cresceu 6%.

A Tabela 5.7 é outra saída do GRASP, ela apresenta a configuração a ser utilizada pelo administrador do sistema para obter o menor custo encontrado para cada SLA, mostrado na Figura 5.7. Por exemplo, para conseguir o custo minimizado para o SLA de 45 segundos, a configuração do sistema deve possuir: o tipo de VM t2.micro; usar 1 instância reservada; o limiar de instanciação 5; o limiar de destruição 1; o número de transcodificações simultâneas deve ser 1; e o **step_size** para VMs elásticas deve ser 1.

A Tabela 5.8 mostra as métricas de cada solução. Para o SLA de 45 segundos, o custo será $ 148,74 (por ano) e o tempo médio de resposta para o usuário 43,53 segundos. Esse sistema também é capaz de realizar 0.099 transcodificações por segundo, e o uso médio de VMs elástica

Tabela 5.7: Solução - Configuração do sistema do Estudo de Caso III

Parâmetro	15 Segundos	30 Segundos	45 Segundos
	Valor	Valor	Valor
Tipo de VM	t2.medium	t2.micro	t2.micro
Instâncias reservadas	2	2	1
Limiar de instanciação	10	10	5
Limiar de destruição	9	8	1
JPVM	1	1	1
step_size	1	1	1

é de 0.61, o que evidencia que uma única instância reservada é insuficiente para atender toda a carga, porém, 2 VMs reservadas trarão desperdício de recurso, devido a isso basta apenas o uso parcial de VMs elásticas para atender os momentos de pico.

Tabela 5.8: Métricas da otimização na nuvem pública - Estudo de Caso III

Métrica	15 Segundos	30 Segundos	45 Segundos
	Valor	Valor	Valor
Uso de Vms Elásticas	3,71E-7	6,651E-4	0,61
Vazão	0,1 transc/s	0,099 transc/s	0,099 transc/s
Tempo Médio de Resposta	13,01 s	29,99 s	43,52 s
Custo	$ 630,71	$ 157,75	$ 148,74

5.4 Estudo de caso IV - Nuvem Privada com Otimização

Este caso de uso apresenta o processo de otimização aplicado às nuvens privadas. Nesse tipo de aplicação, a quantidade de parâmetros e restrições existentes são maiores que os da nuvem pública. Iremos considerar uma carga de trabalho esperada para o sistema composta por vídeos de diferentes tamanhos apresentada na Seção 4.4. Também iremos levar em conta a utilização de 3 servidores iguais, com 4 núcleos e 32 GB de RAM que permitirão instanciar o número de VMs de cada tipo apresentado na Tabela 5.9.

Tabela 5.9: Quantidade de VMs suportada nos servidores

Tipo de VM	Quantidade de VMs suportadas
t2.micro	4
t2.small	4
t2.medium	2
t2.large	2

A otimização tem o objetivo de identificar uma configuração que minimize o custo e respeite o SLA. A configuração do resultado otimizado deve apresentar os parâmetros: quantidade de servidores ligados, limiar para ligar um servidor; limiar para desligar um servidor; tipo de

VM utilizada; quantidade mínima de VMs ligadas; quantidade de trabalhos simultâneos por VM; limiar de instanciação de VM; limiar de destruição de VM; e *step_size*.

Como as transições temporizadas do tempo de transcodificação e tempo de instanciação dependem do tipo de VM e da quantidade de trabalhos simultâneos, necessitamos medir esses valores para as VMs estudadas em um sistema real. Utilizamos a infraestrutura apresentada na Seção 4.4 para medir em cada tipo de VM os tempos de transcodificação para 1, 3, 5, 7, 9 e 11 **JPVM**, que foram obtidos com 60 requisições de transcodificações intervalados de 1 minuto. Os outros valores de **JPVM** foram obtidos por regressão linear para cada tipo de VM, que obteve o R^2 ajustado entre 99,48 % e 99,84 %. Os pontos e retas podem ser vistos na Figura 5.8 e os valores da regressão podem ser vistos na Tabela 5.10.

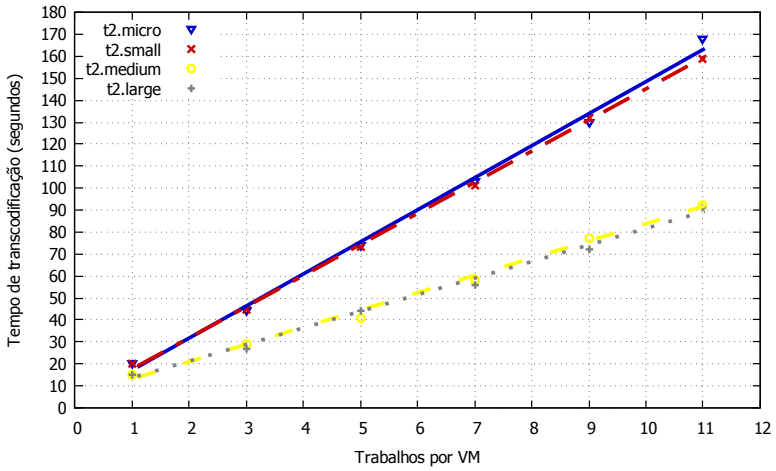


Figura 5.8: Tempo de transcodificação da Arquitetura 4.8 para **JPVM**

Tabela 5.10: Tempo para transcodificação na Arquitetura de Nuvem Privada 4.4

JPVM	**t2.micro (s)**	**t2.small (s)**	**t2.medium (s)**	**t2.large (s)**
1	17,213	17,746	13,216	13,758
2	31,796	31,902	21,062	21,306
3	46,379	46,058	28,908	28,854
4	60,962	60,214	36,754	36,402
5	75,545	74,37	44,6	43,95
6	90,128	88,526	52,446	51,498
7	104,711	102,682	60,292	59,046
8	119,294	116,838	68,138	66,594
9	133,877	130,994	75,984	74,142
10	148,46	145,15	83,83	81,69
11	163,043	159,306	91,676	89,238

Também é necessário o tempo para instanciar uma nova VM de cada tipo, os tempos foram medidos com o algoritmo apresentado no Anexo D, onde foram requisitadas 60 VMs de cada tipo, e medidos os tempos para instanciação da VM com os sistemas de transcodificação. Os valores dos tempos de instanciação podem ser vistos na Tabela 5.11. Também serão

necessários, os tempos para ligar e desligar um servidor, utilizaremos os valores da Tabela 4.13 e 4.14, respectivamente. É importante enfatizar que consideraremos todos esses tempos como exponencialmente distribuídos para realizarmos análises numéricas.

Tabela 5.11: Tempo de instanciação em infraestrutura privada

Tipo de VM	Tempo de Instanciação
t2.micro	24,24 segundos
t2.small	27,13 segundos
t2.medium	32,50 segundos
t2.large	35,66 segundos

O modelo de nuvem privada também requer as medições elétricas, tanto as decorrentes dos servidores, quanto as da utilização de VMs. Nós utilizaremos os valores medidos da infraestrutura da validação da nuvem privada. Então, a potência elétrica de manter um servidor ligado e desligado podem ser vistos na Tabela 4.15. A energia consumida para ligar e desligar um servidor pode ser vista na Tabela 4.16. Já a energia consumida para ligar uma VM irá depender do tipo de VM escolhida, usamos o mesmo método descrito para obtenção da energia consumida para ligar uma VM na Seção 4.4.2 para cada tipo de VM, e seus valores podem ser vistos na Tabela 5.12.

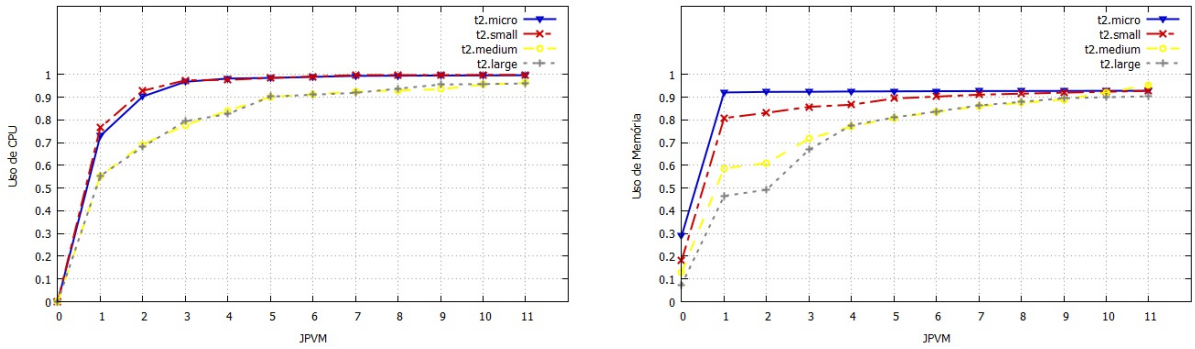
Tabela 5.12: Consumo elétrico para ligar uma VM

Tipo de VM	Consumo elétrico para ligar
t2.micro	0,126306 Wh
t2.small	0,151381 Wh
t2.medium	0,202545 Wh
t2.large	0,229436 Wh

Outra informação importante é a potência consumida durante a transcodificação, essa informação é dada pela identificação dos pesos da Equação 2.4 para cada tipo de VM. Ela necessita da relação entre a quantidade de vídeos transcodificados simultaneamente e uso de CPU e memória. Para isso, monitoramos o uso desses recursos durante a transcodificação, o comportamento desses recursos por cada tipo de VM pode ser visto nas Figuras 5.9(a) e 5.9(b). Após isso, também é necessário identificar a função entre uso de CPU, memória e potência, para isso utilizamos mesmo procedimento usado na Seção 4.4.2, e obtivemos os pesos das equações, que são apresentadas na Tabela 5.13 para cada tipo de VM.

Tabela 5.13: Equações de potência por tipo de VM

Tipo de VM	Equações de potência
t2.micro	$P_{VM}(x) = 20,21 \times u_{cpu}(x) + 1,259 \times u_{mem}(x) + 1,429$
t2.small	$P_{VM}(x) = 20.012 \times u_{cpu}(x) + 0.649 \times u_{mem}(x) + 1.241$
t2.medium	$P_{VM}(x) = 32,245 \times u_{cpu}(x) + 0,85 \times u_{mem}(x) + 3,379$
t2.large	$P_{VM}(x) = 33,298 \times u_{cpu}(x) + 2,573 \times u_{mem}(x) + 2,56$



(a) Uso do CPU por JPVM

(b) Uso da memória por JPVM

Figura 5.9: Uso dos recursos por quantidade de trabalhos simultâneos

Devemos também impor restrições para manter o modelo válido enquanto novas configurações são geradas pelas fases do GRASP. Essas restrições visam evitar que sejam adicionadas quantidades de *tokens* inválidas, por exemplo, que o sistema inicie com mais VMs ligadas do que a capacidade disponível pelos servidores, ou que os limiares gerem condições em que o primeiro disparo é uma transição imediata.

Com essas informações, podemos executar a otimização do modelo dentro dos intervalos contemplados por essas medições. Como discutido na Seção 4.5.1, na otimização o modelo será avaliado por análise estacionária diversas vezes em configurações diferentes. Devido às características da métrica de consumo elétrico do modelo, o algoritmo do anexo B deve ser executado a cada nova configuração na fase de construção e busca local e inserido no modelo antes da execução da análise estacionária, isso foi executado dinamicamente através da API do Mercury nos algoritmos de otimização em Java.

Então, executamos o GRASP considerando os 5 diferentes SLAs da Tabela 5.14, e tempos entre chegadas de 10 segundos exponencialmente distribuídos. Também definimos os intervalos de busca de cada parâmetro: quantidade de servidores ligados no início de 1 até a quantidade máxima de servidores; limiar de instanciação de um servidor, de 0 até 10; limiar para desligar um servidor, de 1 até 10; A quantidade de VMs ligadas no início da execução, de 1 até a capacidade máxima oferecida pelos servidores ligados no início; limiar de instanciação, no intervalo 1 até 10; limiar de destruição, de 1 até 10; *step_size*, no intervalo 1 até 10; e a quantidade de trabalhos simultâneos, variando de 1 até 10.

Os consumos elétricos consideram um ano de utilização das VMs transcodificadoras e foram calculados em kWh. O comportamento do menor consumo encontrado na otimização para cada SLA é retratado na Figura 5.10. Como na nuvem pública, os custos diminuem com o aumento do SLA de tempo médio de resposta. Podemos ver também que os custos não crescem linearmente com o SLA, uma vez que reduzir 15 segundos do SLA de 30 segundos, leva a um aumento de mais de 46% no consumo elétrico, enquanto que o aumento de 15 segundos reduz apenas 5% da energia necessária para cumprir o SLA. Uma pequena variação também aparece entre SLAs de 45, 60 e 75, nesses intervalos o administrador pode utilizar SLAs mais restritivos

Tabela 5.14: Restrições (SLAs) em nuvens privadas

Caso	Medida	Valor
1	Vazão mínima	0,097 transc/s
	Tempo máximo de resposta	15 segundos
2	Vazão mínima	0,097 transc/s
	Tempo máximo de resposta	30 segundos
3	Vazão mínima	0,097 transc/s
	Tempo máximo de resposta	45 segundos
4	Vazão mínima	0,097 transc/s
	Tempo máximo de resposta	60 segundos
5	Vazão mínima	0,097 transc/s
	Tempo máximo de resposta	75 segundos

sem que haja um grande impacto no consumo.

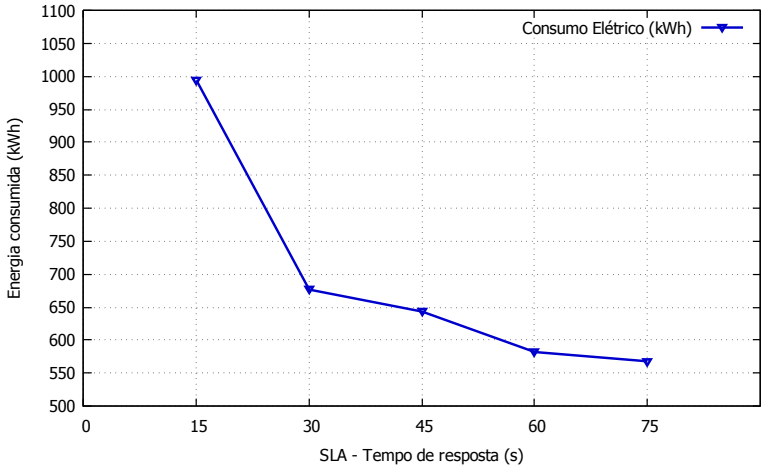


Figura 5.10: Consumo elétrico por tempo de resposta máximo

Outra característica encontrada nos resultados é a diferença entre o SLA e o tempo médio de resposta encontrado em alguns resultados, como pode ser observado pela Tabela 5.15, que possui as métricas encontradas. Essa diferença tem origem nos custos de ligar uma VM, pois instanciar e destruir uma VM inúmeras vezes, para atender apenas parte do pico de requisições, traz também uma maior vazão de instanciação, o que aumenta o custo total.

Tabela 5.15: Métricas da otimização - nuvem privada

Métrica	15 Seg	30 Seg	45 Seg	60 Seg	75 Seg
	Valor	Valor	Valor	Valor	Valor
Vazão	0,1 trans/s	0,1 trans/s	0,1 trans/s	0,1 trans/s	0,099 trans/s
T. Méd. de Resp.	14,77 s	22,74 s	43.25 s	59.36 s	67.96 s
Consumo elétrico	994,0 kWh	676,5 kWh	643,3 kWh	581,8 kWh	567,8 kWh

Por fim, a Tabela 5.16 apresenta a configuração que deve ser executada em cada SLA, para obtenção das métricas da Tabela 5.15. Para o SLA de 15 segundos, a configuração requer: o

uso da VM do tipo t2.medium; pelo menos 2 servidores sempre ligados, o que significa que o custo de manter um único servidor ligado e ligar um segundo de acordo com a demanda apenas aumentará o custo; o limiar para ligar um novo servidor é 0, apenas quando não houver mais capacidade de instanciação de VMs deve-se ligar um servidor; para desligar o nó apenas quando houver a capacidade suficiente para ligar 4 VMs; a quantidade de VMs instanciadas no início do sistema é 3; o limiar para a instanciação de VMs é 10; o limiar para destruição 3; o *stepsize* é 1; e a quantidade de trabalhos simultâneos por VM (JPVM) é 1.

Tabela 5.16: Configuração otimizada - nuvem privada

Parâmetro	**15 Seg**	**30 Seg**	**45 Seg**	**60 Seg**	**70 Seg**
	Valor	**Valor**	**Valor**	**Valor**	**Valor**
NODE_ON_START	2	1	1	1	1
ThresholdStartNode	0	0	0	0	0
ThresholdShutdownNode	4	6	6	6	7
Tipo de VM	t2.medium	t2.small	t2.micro	t2.micro	t2.micro
VM_{ON}	3	3	2	1	1
THR_INST	10	10	6	8	10
THR_DEST	1	2	1	1	1
JPVM	1	1	1	1	1
Step_Size	1	1	1	2	2

5.5 Estudo de caso V - Tempo para encontrar soluções otimizadas

Em muitas situações reais serão necessárias grandes quantidades de VMs para suportar a demanda que cumpra um SLA, então para modelar esses casos deverá haver um aumento dos *tokens* nos locais que representam recursos nos modelos das Figuras 4.1 e 4.2. Isso irá aumentar o tempo necessário para computar as métricas de interesse, e como consequência dificultará as aplicações dos modelos para uma solução otimizada (Casos de Uso III e IV), pois elas requerem inúmeras execuções do modelo. Para isso, apresentamos 3 variações do algoritmo de busca local com intuito de reduzir o tempo para encontrar boas soluções otimizadas.

O último caso de uso apresenta uma comparação do tempo necessário para encontrar soluções adequadas através das diversas variações dos algoritmos de otimização da Seção 4.5. Executamos a otimização do modelo de nuvem pública com os algoritmos de busca local 6, 7 e 8, em um servidor HP Proliant DL320e G8 com processador Intel Xeon E3-1240v2 (3.40 GHz e 8 MB Cache), e memória RAM de 16 GB, com uso do Mercury por análise numérica pelo método de GTH.

Executamos o algoritmo com os seguintes parâmetros: o tamanho RLC definido por $\alpha = 0,7$; o intervalo de variação dos elementos de cada parâmetro $\beta = 0,5$; a variação da vizinhança de cada valor é $\gamma = 0,1$; e a quantidade máxima de iterações na busca local são 40 buscas. O SLA de tempo médio de resposta é de 50 segundos, enquanto que o de vazão é de 0,098, a carga do sistema é composta de tempos entre requisições de 10 segundos.

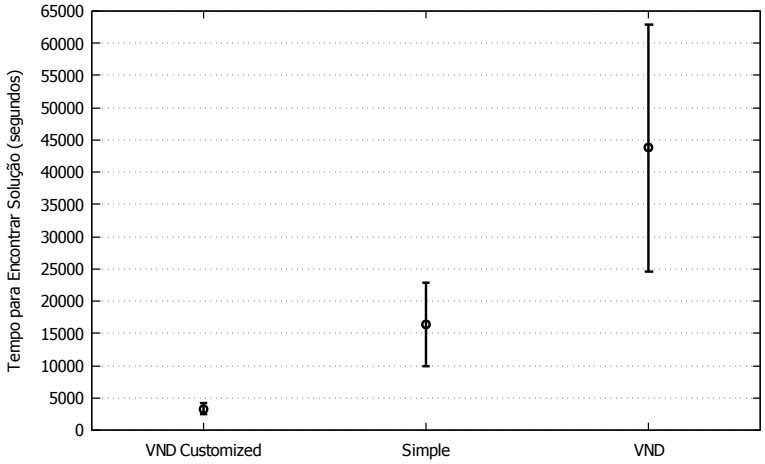


Figura 5.11: Tempo para encontrar uma solução por método de busca local.

Utilizamos também uma solução (S) previamente calculada como parte do critério de parada, onde o procedimento termina quando é encontrada uma solução com custo menor que $S \times 1,05$ ou é alcançado o número máximo de iterações (20000).

A otimização foi executada 30 vezes para cada algoritmo e os resultados dos tempos de resposta são normais com 95% de confiança. A Figura 5.11 apresenta os resultados dos tempos médios para encontrar uma solução. Podemos ver que o método de menor tempo para encontrar resposta é o VND customizado com tempo médio de 55 minutos para encontrar a resposta, seguido pelo método simples com 4 horas e 30 minutos, e por fim o método VND com 12 horas.

Esses resultados demonstram a vantagem de nossa abordagem, cujo tempo necessário para encontrar a solução representa 7,5% do tempo do VND e 20% do método simples. Também pode ser observado na Figura 5.11 que os intervalos de confiança não se cruzam, o que evidencia que os tempos para encontrar a resposta são estatisticamente diferentes (JAIN, 1990).

5.6 Considerações Finais

Este capítulo apresentou a aplicações dos modelos e estratégias propostas, demonstrando questões que podem ser resolvidas através dessas abordagens em cenários comumente encontrados em sistemas reais. Essa abordagem tanto permite que o administrador antecipe o comportamento do desempenho do sistema através da variação individual de cada parâmetro, como também receba uma configuração otimizada que deve ser configurada em uma nuvem pública ou privada para reduzir o custo e cumprir o SLA quando o sistema é submetido a uma carga.

6

Considerações Finais

A transcodificação de vídeo tem sido amplamente utilizada para fornecer vídeos adaptados ao atual ambiente heterogêneo de clientes. Essa atividade consome diversos recursos, e em muitos casos possui períodos de baixa e alta demanda. A computação em nuvem tem sido utilizada para suportar esses sistemas, principalmente devido à característica elástica, que uma vez adequadamente configurada permite ajustar a capacidade de processamento, oferecendo o desempenho acordado com os usuários do sistema, adicionando recursos em momentos de alto volume de requisições e retirando recursos, para evitar desperdício, quando houver baixa demanda.

Entretanto, a configuração adequada do *auto-scaling* não é uma tarefa simples (BISWAS et al., 2015), depende de diversos fatores, que variam de acordo com o tipo de nuvem utilizada, para uma nuvem pública devem ser configurados fatores como: a carga de trabalho esperada; o tempo necessário para transcodificação; o tempo que demora a criação de uma nova instância; o preço de diferentes contratos; e da identificação do momento que deve ser adicionada e retirada uma instância. Em nuvens privadas adiciona-se a configuração da infraestrutura física, com o datacenter *right-size* e o consumo elétrico associado aos servidores e VMs.

Devido à complexidade inerente da configuração adequada, utilizamos modelos SPN para auxiliar na compreensão do resultado da variação desses parâmetros nas métricas de vazão, tempo médio de resposta e custo do sistema. Para a nuvem pública consideramos os custos do tempo de uso de VMs e em nuvem privada o consumo elétrico. Os modelos foram validados com 95% de confiança em relação a cenários reais.

Também propomos a utilização de mecanismos de otimização que exploram o espaço de configurações possíveis do modelo em busca da redução do custo. Portanto, essa dissertação não oferece apenas um modelo para verificação do comportamento de infraestruturas de transcodificação de vídeo em nuvens públicas e privadas, mas também um mecanismo que apresenta quais os valores de cada parâmetro e contratos que devem ser aplicadas na nuvem para que o administrador consiga cumprir os requisitos de desempenho definidos no SLA com um custo otimizado. Além disso, apresentamos uma versão do algoritmo *VND* de busca local para o modelo, que apresenta soluções otimizadas em menor tempo que a versão padrão do *VND*, e

mecanismos de busca local clássico.

Apresentamos 5 casos de uso que demonstram a aplicação do modelo, o primeiro foca no efeito dos parâmetros no tempo médio de resposta e custo em nuvens públicas. O segundo analisa a probabilidade de uma determinada configuração completar um conjunto de trabalhos em um dado tempo.

O terceiro caso utiliza o algoritmo de otimização com três diferentes SLAs, e foi constatado que o acréscimo ou redução de 15 segundos no SLA podem apresentar diferentes variações no custo do sistema, do SLA de 30 para 15 segundos houve um crescimento de 299% do custo, no entanto, apenas um aumento de apenas 6% de 45 para 30 segundos. O mesmo comportamento, porém com uma menor variação, foi observado nas configurações otimizadas de nuvens privadas de 15 para 30 segundos houve um acréscimo de 46 % no consumo elétrico, enquanto que de 45 para 30 segundos, houve apenas um aumento de 5%. Esse tipo de estudo permite que o administrador possua informações adequadas para identificar os custos que cada SLA irá impor ao sistema, e até onde o SLA pode ser ajustado sem que incorra em um grande crescimento no custo do sistema.

Por fim, o estudo de caso 5 apresenta o tempo médio para encontrar soluções em nuvens públicas através de diferentes algoritmos de busca local com o GRASP, o que evidencia a eficiência da nossa estratégia de otimização para modelos SPN, que leva 7,5% do tempo da busca VND, e 20% da busca simples.

6.1 Contribuições

Como resultado das atividades desenvolvidas neste trabalho, podemos identificar as seguintes contribuições:

- Elaboração de um modelo de infraestrutura publica de transcodificação de vídeo: foram desenvolvidos modelos validados com 95% de confiança, que permitem identificar o custo do aluguel e o desempenho de diferentes configurações de *auto-scaling* em nuvem.

- Elaboração de um modelo para infraestrutura privada de transcodificação de vídeo: combinamos modelos de consumo elétrico baseados na utilização de recursos com modelos estocásticos para obter o consumo elétrico em nuvens privadas, baseando-se: na quantidade de servidores ligados; no consumo elétrico de VMs; e na carga de trabalho.

- Utilização do algoritmo de otimização GRASP em modelos: utilizamos os modelos gerados para encontrar uma configuração que respeite o SLA e minimize o custo do aluguel em nuvens públicas ou consumo elétrico em nuvens privadas.

- Desenvolvimento de um algoritmo de busca local: apresentamos uma variação do algoritmo de busca local VND que apresenta uma redução considerável no tempo para encontrar soluções para os modelos propostos.

- Identificação do comportamento do custo em relação ao SLA: os resultados dos casos de uso de otimização demonstram que uma mesma variação no tempo médio de resposta apresenta variações muito grandes no custo. Evidenciando-se a escolha adequada do SLA como um elemento chave na redução dos custos do sistema.

6.2 Limitações e Trabalhos Futuros

Essa pesquisa apresenta diversas possibilidade de extensão para atividades futuras, das quais podemos citar:

- A utilização de diferentes tipos de VM: atualmente o modelo representa apenas a utilização de um tipo de VM por execução, a utilização de mais de um tipo de VM pode permitir uma redução ainda maior do custo nas diferentes infraestruturas, ajustando o sistema com maior flexibilidade a carga de trabalho.

- Os modelos de nuvem privada consideraram apenas um tipo de servidor: nos modelos de nuvem privada foram apenas considerados servidores com a mesma quantidade de recursos e sempre o mesmo mecanismo para ligar e desligar. Podem ser adicionados ao modelo a utilização de uma infraestrutura mista com capacidades e consumos diferentes. Além de também incluir custos de cada *hardware* diferente, juntamente com o custo de propriedade dessa infraestrutura privada.

- Desenvolvimento de algoritmos de otimização mais eficientes, além de um estudo mais aprofundado na busca de configurações melhores em algoritmos de otimização aplicados a modelagem estocástica.

- Os modelos utilizados neste trabalho apresentam estruturas isoladas, ou seja, utilizando apenas uma infraestrutura pública ou privada, podem ser desenvolvidas estratégias de combinação desses modelos para dar origem a uma infraestrutura híbrida, que busque reduzir o custo de operação do sistema de transcodificação.

- Os modelos propostos de nuvem elástica consideraram apenas aplicações de transcodificação de vídeo, que são bastante intensivas em uso de CPU e memória. Podem ser realizados estudos identificando outros tipos de sistemas que utilizem a capacidade elástica da nuvem, e então, adequar essa estratégia a um contexto mais amplo.

- As métricas utilizadas nos modelos apenas se concentraram em desempenho e custo. Outro trabalho futuro poderia abordar a utilização de métricas de performabilidade, com objetivo dos mecanismos de *auto-scaling* também desempenharem o papel de garantir um desempenho aceitável na ocorrência de falhas e, consequentemente, o custo associado a essa garantia.

Referências

AHMAD, I. et al. Video transcoding: an overview of various techniques and research issues. **Multimedia, IEEE Transactions on**, [S.l.], v.7, n.5, p.793–804, 2005.

AJMONE MARSAN, M.; CONTE, G.; BALBO, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. **ACM Transactions on Computer Systems (TOCS)**, [S.l.], v.2, n.2, p.93–122, 1984.

AMAZON. **AWS Case Study**: encoding. (Accessed on 11/16/2016), `https://aws.amazon.com/pt/solutions/case-studies/encoding/`.

ANDERSON, T. W.; DARLING, D. A. A test of goodness of fit. **Journal of the American statistical association**, [S.l.], v.49, n.268, p.765–769, 1954.

APARICIO-PARDO, R. et al. Transcoding live adaptive video streams at a massive scale in the cloud. In: ACM MULTIMEDIA SYSTEMS CONFERENCE, 6. **Proceedings...** [S.l.: s.n.], 2015. p.49–60.

ASSUNÇÃO, M. D. de et al. Impact of user patience on auto-scaling resource capacity for cloud services. **Future Generation Computer Systems**, [S.l.], v.55, p.41–50, 2016.

AWS. **Amazon EC2 Pricing**. (Accessed on 11/26/2016), `https://aws.amazon.com/ec2/pricing/`.

BISWAS, A. et al. An Auto-Scaling Framework for Controlling Enterprise Resources on Clouds. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2015 15TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2015. p.971–980.

BOHRA, A. E. H.; CHAUDHARY, V. VMeter: power modelling for virtualized clouds. In: PARALLEL & DISTRIBUTED PROCESSING, WORKSHOPS AND PHD FORUM (IPDPSW), 2010 IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.1–8.

BOLCH, G. et al. **Queueing networks and Markov chains**: modeling and performance evaluation with computer science applications. [S.l.]: John Wiley & Sons, 2006.

BRINKSMA, E.; HERMANNS, H.; KATOEN, J.-P. **Lectures on Formal Methods and Performance Analysis**: summer school on trends in computer science, the netherlands, july 3-7, 2000. revised lectures. [S.l.]: Springer, 2003. v.2090.

BURKE, E. K.; KENDALL, G. et al. **Search methodologies**. [S.l.]: Springer, 2005.

CAMPOS, E. et al. Performance Evaluation of Virtual Machines Instantiation in a Private Cloud. In: SERVICES (SERVICES), 2015 IEEE WORLD CONGRESS ON. **Anais...** [S.l.: s.n.], 2015. p.319–326.

CHAKRAVARTI, I. M.; LAHA, R. G. Handbook of methods of applied statistics. In: **Handbook of methods of applied statistics**. [S.l.]: John Wiley & Sons, 1967.

CHENG, X.; LIU, J.; DALE, C. Understanding the characteristics of internet short video sharing: a youtube-based measurement study. **Multimedia, IEEE Transactions on**, [S.l.], v.15, n.5, p.1184–1194, 2013.

CHUNG, C. A. **Simulation modeling handbook**: a practical approach. [S.l.]: CRC press, 2003.

CISCO SYSTEMS. **White paper**: cisco visual network index forecast and methodology, 2015-2020 - cisco. (Accessed on 11/14/2016), `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

CLOUDSTACK. **Apache CloudStack**: open source cloud computing. (Accessed on 12/03/2016), `https://cloudstack.apache.org/`.

COLMENAR, J. M. et al. Advanced Greedy Randomized Adaptive Search Procedure for the Obnoxious p-Median problem. **European Journal of Operational Research**, [S.l.], v.252, n.2, p.432–442, 2016.

CPULIMIT. **CPU limit**. (Accessed on 12/06/2016), `http://cpulimit.sourceforge.net/`.

DE SOUSA, E. T. G. et al. Performance and Cost Modeling Strategy for Cloud Infrastructure Planning. In: IEEE 7TH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, 2014. **Anais...** [S.l.: s.n.], 2014. p.546–553.

DUPONT, S. et al. Experimental analysis on autonomic strategies for cloud elasticity. In: CLOUD AND AUTONOMIC COMPUTING (ICCAC), 2015 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2015. p.81–92.

ENCODING.COM. **Cloud based Transcoding | Encoding.com**. (Accessed on 11/16/2016), `https://www.encoding.com/`.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. **Operations research letters**, [S.l.], v.8, n.2, p.67–71, 1989.

FESTA, P.; RESENDE, M. G. GRASP: an annotated bibliography. In: **Essays and surveys in metaheuristics**. [S.l.]: Springer, 2002. p.325–367.

FFmpeg. (Accessed on 11/22/2016), `https://www.ffmpeg.org/`.

GALANTE, G.; BONA, L. C. E. de. A survey on cloud computing elasticity. In: UTILITY AND CLOUD COMPUTING (UCC), 2012 IEEE FIFTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.263–270.

GENDREAU, M.; POTVIN, J.-Y. **Handbook of metaheuristics**. [S.l.]: Springer, 2010. v.2.

GERMAN, R. **Performance analysis of communication systems with non-Markovian stochastic Petri nets**. [S.l.]: John Wiley & Sons, Inc., 2000.

GHANBARI, H. et al. Exploring alternative approaches to implement an elasticity policy. In: CLOUD COMPUTING (CLOUD), 2011 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.716–723.

GU, C.; HUANG, H.; JIA, X. Power metering for virtual machine in cloud computing-challenges and opportunities. **IEEE Access**, [S.l.], v.2, p.1106–1116, 2014.

HAVERKORT, B. R. Markovian models for performance and dependability evaluation. In: **Lectures on Formal Methods and PerformanceAnalysis**. [S.l.]: Springer, 2001. p.38–83.

HERNÁNDEZ-PÉREZ, H.; RODRÍGUEZ-MARTÍN, I.; SALAZAR-GONZÁLEZ, J. J. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. **Computers & Operations Research**, [S.l.], v.36, n.5, p.1639–1645, 2009.

IQBAL, W. et al. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. **Future Generation Computer Systems**, [S.l.], v.27, n.6, p.871–879, 2011.

JAIN, R. **The art of computer systems performance analysis**: techniques for experimental design, measurement, simulation, and modeling. [S.l.]: John Wiley & Sons, 1990.

JMETER. **Apache JMeter - Apache JMeter™**. (Accessed on 11/23/2016), `http://jmeter.apache.org`.

KANSAL, A. et al. Virtual machine power metering and provisioning. In: ACM SYMPOSIUM ON CLOUD COMPUTING, 1. **Proceedings...** [S.l.: s.n.], 2010. p.39–50.

KLEINROCK, L. Queueing systems, volume I: theory. **,** [S.l.], 1975.

KRISHNAPPA, D. K.; ZINK, M.; SITARAMAN, R. K. Optimizing the video transcoding workflow in content delivery networks. In: ACM MULTIMEDIA SYSTEMS CONFERENCE, 6. **Proceedings...** [S.l.: s.n.], 2015. p.37–48.

LAO, F.; ZHANG, X.; GUO, Z. Parallelizing video transcoding using map-reduce-based cloud computing. In: CIRCUITS AND SYSTEMS (ISCAS), 2012 IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2012. p.2905–2908.

LI, Y. et al. An online power metering model for cloud environment. In: NETWORK COMPUTING AND APPLICATIONS (NCA), 2012 11TH IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2012. p.175–180.

LIN, C.-C.; LIU, P.; WU, J.-J. Energy-efficient virtual machine provision algorithms for cloud systems. In: UTILITY AND CLOUD COMPUTING (UCC), 2011 FOURTH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.81–88.

LIN, M. et al. Dynamic right-sizing for power-proportional data centers. **IEEE/ACM Transactions on Networking (TON)**, [S.l.], v.21, n.5, p.1378–1391, 2013.

LIN, S. et al. Parallelizing video transcoding with load balancing on cloud computing. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS2013), 2013. **Anais...** [S.l.: s.n.], 2013. p.2864–2867.

LITTLE, J. D. A proof for the queuing formula: l= λ w. **Operations research**, [S.l.], v.9, n.3, p.383–387, 1961.

LORIDO-BOTRÁN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. Auto-scaling techniques for elastic applications in cloud environments. **Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09**, [S.l.], v.12, p.2012, 2012.

MARSAN, M. A. Stochastic Petri nets: an elementary introduction. In: EUROPEAN WORKSHOP ON APPLICATIONS AND THEORY IN PETRI NETS. **Anais...** [S.l.: s.n.], 1988. p.1–29.

MARSAN, M. A. et al. **Modelling with generalized stochastic Petri nets**. [S.l.]: John Wiley & Sons, Inc., 1994.

MARSAN, M. A. et al. Modelling with generalized stochastic petri nets. **ACM SIGMETRICS Performance Evaluation Review**, [S.l.], v.26, n.2, p.2, 1998.

MATOS, R.; MACIEL, P. R.; SILVA, R. M. Sensitive GRASP: combinatorial optimisation of composite web services guided by sensitivity analysis. **International Journal of Web and Grid Services**, [S.l.], v.12, n.1, p.63–80, 2016.

MELL, P. M.; GRANCE, T. **The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States: [s.n.], 2011.

MERLIN, P.; FARBER, D. Recoverability of communication protocols-implications of a theoretical study. **IEEE transactions on Communications**, [S.l.], v.24, n.9, p.1036–1043, 1976.

MÖBIUS, C.; DARGIE, W.; SCHILL, A. Power consumption estimation models for processors, virtual machines, and servers. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.], v.25, n.6, p.1600–1614, 2014.

MOLLOY, M. K. On the integration of delay and throughput measures in distributed processing models. **,**, [S.l.], 1981.

MURATA, T. Petri nets: properties, analysis and applications. **Proceedings of the IEEE**, [S.l.], v.77, n.4, p.541–580, 1989.

NMON for Linux. (Accessed on 11/23/2016), `http://nmon.sourceforge.net/pmwiki.php`.

NOE, J. D.; NUTT, G. J. Macro E-nets for representation of parallel systems. **IEEE Transactions on Computers**, [S.l.], v.100, n.8, p.718–727, 1973.

PETRI, C. Kommunikation mit Automaten Bonn: institut für instrumentelle mathematik. **Schriften des IIM Nr**, [S.l.], v.2, 1962.

PRAIS, M.; RIBEIRO, C. **Parameter variation in grasp procedures. Investigacion Operativa Vol. 9 (2000) 1-20. 10. Reinelt, G.**: the linear ordering problem: algorithms and applications. [S.l.]: berlin: Heldermann verlag. Mathematical Social Sciences, 1985.

RAMCHANDANI, C. **ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS**. Cambridge, MA, USA: [s.n.], 1974.

RIBAS, M. et al. Modeling the use of spot instances for cost reduction in cloud computing adoption using a Petri net framework. In: INTEGRATED NETWORK MANAGEMENT (IM), 2015 IFIP/IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2015. p.1428–1433.

SALEHIPOUR, A. et al. Efficient GRASP+ VND and GRASP+ VNS metaheuristics for the traveling repairman problem. **4OR**, [S.l.], v.9, n.2, p.189–209, 2011.

SHARMA, O.; SAINI, H. VM Consolidation for Cloud Data Center Using Median Based Threshold Approach. **Procedia Computer Science**, [S.l.], v.89, p.27–33, 2016.

SILVA, B. et al. Mercury: an integrated environment for performance and dependability evaluation of general systems. In: INDUSTRIAL TRACK AT 45TH DEPENDABLE SYSTEMS AND NETWORKS CONFERENCE (DSN). **Proceedings...** [S.l.: s.n.], 2015.

STOCKHAMMER, T. Dynamic adaptive streaming over HTTP–: standards and design principles. In: ACM CONFERENCE ON MULTIMEDIA SYSTEMS. **Proceedings...** [S.l.: s.n.], 2011. p.133–144.

STRESS. **stress(1)**: impose load on/stress test systems - linux man page. (Accessed on 12/06/2016), `https://linux.die.net/man/1/stress`.

TIMMERER, C. et al. Transcoding and streaming-as-a-service for improved video quality on the web. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA SYSTEMS, 7. **Proceedings...** [S.l.: s.n.], 2016. p.37.

TRIVEDI, K. S. **Probability & statistics with reliability, queuing and computer science applications**. [S.l.]: John Wiley & Sons, 2008.

TUFFIN, B. et al. Simulation versus analytic-numeric methods: a petri net example. In: VALUETOOLS CONFERENCE, 2. **Proceedings...** [S.l.: s.n.], 2007.

VERSICK, D.; WASSMANN, I.; TAVANGARIAN, D. Power consumption estimation of CPU and peripheral components in virtual machines. **ACM SIGAPP Applied Computing Review**, [S.l.], v.13, n.3, p.17–25, 2013.

W3C. **World Wide Web Consortium Towards Video on the Web with HTML5**. (Accessed on 11/14/2016), `https://www.w3.org/2010/Talks/1014-html5-video-fd/pdf/slides.pdf`.

W3SCHOOLS. **HTML5 Video**. (Accessed on 11/14/2016), `http://www.w3schools.com/html/html5_video.asp`.

WADIA, Y. **AWS Administration–The Definitive Guide**. [S.l.]: Packt Publishing Ltd, 2016.

WATTS up. (Accessed on 11/23/2016), `https://www.wattsupmeters.com/secure/index.php`.

WU, W.; LIN, W.; PENG, Z. An intelligent power consumption model for virtual machines under CPU-intensive workload in cloud environment. **Soft Computing**, [S.l.], p.1–10, 2016.

XENSERVER. **Xenserver Open Source Server Virtualization**. (Accessed on 12/02/2016), `http://xenserver.org/overview-xenserver-open-source-virtualization/download.html`.

XIAO, Z.; CHEN, Q.; LUO, H. Automatic scaling of internet applications for cloud computing services. **Computers, IEEE Transactions on**, [S.l.], v.63, n.5, p.1111–1123, 2014.

YANG, H. et al. iMeter: an integrated vm power model based on performance profiling. **Future Generation Computer Systems**, [S.l.], v.36, p.267–286, 2014.

YANG, M. et al. Adaptive configuration of cloud video transcoding. In: CIRCUITS AND SYSTEMS (ISCAS), 2015 IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2015. p.1658–1661.

YOUTUBE. **Statistics - YouTube**. (Accessed on 11/14/2016), `https://www.youtube.com/yt/press/statistics.html`.

Apêndices

A

Medição Elétrica - Watts Up Meter

Utilizamos o equipamento externo *Watts Up* para medir a potência e o consumo elétrico diretamente dos servidores, esse equipamento permite medir e registrar a potência elétrica com a precisão de +/- 1,5%, com o intervalo de medição de no mínimo 1 segundo. Nossa arquitetura de medição é retratada na Figura A.1, onde os dois servidores são ligados a um multiplicador de tomadas, para medição na saída de potência do *Watts UP* que é ligado na tomada elétrica.

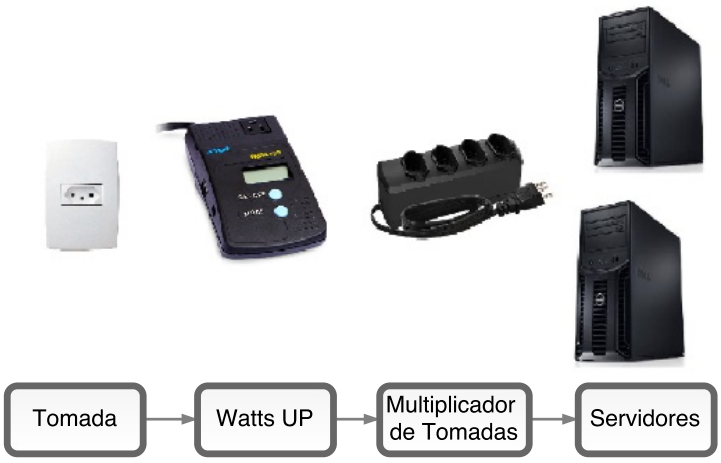


Figura A.1: Arquitetura Medidor de Potência.

As medições são gravadas na memória interna do dispositivo, que permitem até 4000 registros com todas as 18 métricas possíveis, ou 120.000, com a seleção de apenas uma. Os registros são obtidos por meio de um *software* que recupera os registros de acordo com a configuração de tempo e métricas predefinidas. O equipamento também permite a sincronização do tempo com o computador que recebe os dados. Então configuramos o computador e as VMs que monitoram os recursos das VMs através do mesmo servidor de tempo de rede (*Network Time Protocol - NTP*).

B

Métrica de Consumo Elétrico da VM

O Algoritmo *Econsume* em Java gera a métrica de consumo elétrico das VMs na sintaxe do *software* Mercury, ele recebe como entradas, a quantidade de servidores *sizeInfra*, tipo de VM *vmType*, a quantidade de VMs por servidor *nodeVM*, e a quantidade de trabalhos simultâneos *JPVM*. Esse algoritmo também utiliza a função *eCost* para gerar os valores de custo elétrico da infraestrutura. É importante notar que o resultado é uma lista de *strings*, cada elemento dessa lista deve ser uma métrica do Mercury.

```
1   public ArrayList<String> Econsume(int sizeInfra,String vmType, int
    nodeVM, int JPVM ) {
2       int n = nodeVM * sizeInfra + 1;
3       String metric = "";
4       String vms = "";
5       ArrayList<String> result = new ArrayList<>();
6       int maxEQ = 7;
7       for (int i = 1; i < n; i++) {
8           vms = "((P{((#ARes+#TJ+#Rrel)/" + JPVM + ")=" + i + "})*(";
9           metric = vms;
10          for (int j = 0; j <= i * JPVM; j++) {
11              metric = metric + "((P{#TJobs=" + j + "})*(";
12              for (int k = j; k < (j + i); k++) {
13                  metric = metric + "(" + eCost((int) Math.ceil(k / i),
    vmType) + ")";
14                  if (k < (j + i - 1)) {
15                      metric += "+";
16                  }
17              }
18              metric = metric + "))";
19              if (j < i * JPVM && (j \% maxEQ != 0) || (j == 0)) {
20                  metric = metric + "+";
21              } else if (j != 0 && j \% maxEQ == 0) {
22                  metric = metric + "))";
23                  result.add(metric);
24                  if (j != i * JPVM) {
25                      metric = vms;
```

```
                    } else {
                        metric = "";
                    }
                }
            }
            if (!metric.isEmpty()) {
                metric = metric + "))";

                result.add(metric);
            }
        }
        return result;
        }
```

O algoritmo *eCost* utiliza uma lista com a utilização de CPU e memória juntamente com as equações de potência elétrica, esses valores são os valores medidos na infraestrutura apresentada na seção 4.2.

```
    public double eCost(int transcJobs, String vmType) {

     HashMap<String, ArrayList<Double>> jobsToMEM = new HashMap<String,
ArrayList<Double>>() {
         {
             put("t2.micro", new ArrayList<>(Arrays.asList(0.289746094,
0.920888672, 0.923515625, 0.923964844, 0.925332031, 0.925839844,
0.926474609, 0.927109375, 0.927363281, 0.927617188, 0.928046875,
0.928476563)));
             put("t2.small", new ArrayList<>(Arrays.asList(0.180957031,
0.808032227, 0.831152344, 0.857407227, 0.866855469, 0.894355469,
0.902644043, 0.910932617, 0.915527344, 0.92012207, 0.923979492,
0.927832031)));
             put("t2.medium", new ArrayList<>(Arrays.asList(0.130029297,
 0.586977539, 0.609645996, 0.71729248, 0.770566406, 0.809116211,
0.834602051, 0.860085449, 0.87473999, 0.889394531, 0.921175537,
0.952956543)));
             put("t2.large", new ArrayList<>(Arrays.asList(0.073937988,
0.582232422, 0.603652344, 0.741089844, 0.825914063, 0.852105469,
0.872806641, 0.893505859, 0.905994141, 0.918482422, 0.921776367,
0.925070313)));
         }
     };

     HashMap<String, ArrayList<Double>> jobsToCPU = new HashMap<String,
ArrayList<Double>>() {
         {
             put("t2.micro", new ArrayList<>(Arrays.asList(0.0, 0.7292,
0.9033, 0.9679, 0.9823, 0.9851, 0.98955, 0.994, 0.99475, 0.9955, 0.9968)
));
```

```
15                  put("t2.small", new ArrayList<>(Arrays.asList(0.0, 0.766,
    0.9283, 0.9744, 0.9765, 0.9847, 0.99105, 0.9974, 0.9974, 0.9974, 0.9985)
    ));
16                  put("t2.medium", new ArrayList<>(Arrays.asList(0.0, 0.5538,
     0.6921, 0.7768, 0.8405, 0.9014, 0.91305, 0.9247, 0.92985, 0.935,
    0.9634)));
17                  put("t2.large", new ArrayList<>(Arrays.asList(0.0, 0.5534,
    0.6809, 0.7946, 0.8259, 0.9024, 0.91085, 0.9193, 0.9375, 0.9557, 0.9602)
    ));
18              }
19          };
20          double cpu = 0, mem = 0, e = 0;
21          if (vmType.equalsIgnoreCase("t2.micro")) {
22              cpu = 20.21;
23              mem = 1.259;
24              e = 1.429;
25          }
26          if (vmType.equalsIgnoreCase("t2.small")) {
27              cpu = 20.012;
28              mem = 0.649;
29              e = 1.241;
30          }
31          if (vmType.equalsIgnoreCase("t2.medium")) {
32              cpu = 32.245;
33              mem = 0.85;
34              e = 3.379;
35          }
36          if (vmType.equalsIgnoreCase("t2.large")) {
37              cpu = 33.298;
38              mem = 2.573;
39              e = 2.56;
40          }
41          double result = e + cpu * jobsToCPU.get(vmType).get(transcJobs) +
    mem * jobsToMEM.get(vmType).get(transcJobs);
42          return result;
43      }
```

C
Jmeter

Jmeter é um *software* de código aberto, projetado para realizar testes de carga e funcionais. Foi utilizado para obter: os tempos de transcodificação; utilização de recursos da VM; e realizar a validação do modelo. Ele possui diversas funções de controle de tempo, como o campo *Ramp Up Period* da Figura C.1, ou *Constant Timer*, *Random Timer*, entre outras. Porém, não apresenta por padrão uma função adequada para validar taxas de entrada exponenciais, como as necessárias no capítulo 4. Devido a isso utilizamos uma função customizada a partir do uso do *BFS Timer*, e da variável definida pelo usuário *beforeTime* como pode ser visto no algoritmo abaixo.

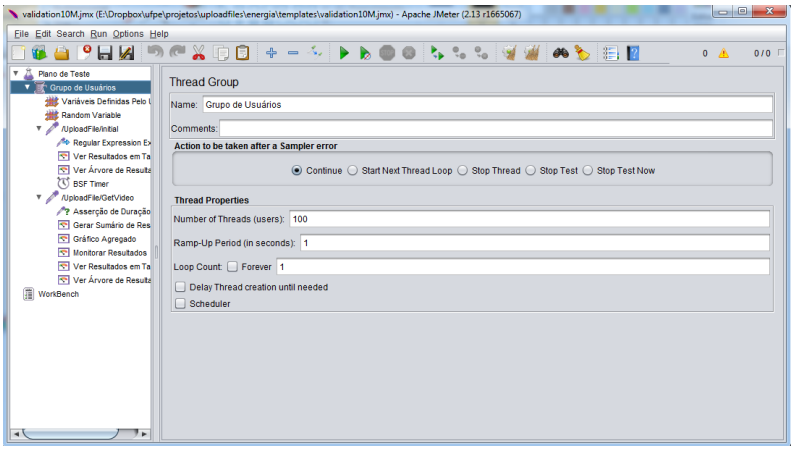


Figura C.1: Configuração de usuários do Jmeter.

```
//o tempo deve ser dado em milisegundos
double lambda = 1.0/10000.0;

loc = props.get("beforeTime");

if(loc == null){
   props.put("beforeTime", "0.0");
}
Double before = Double.parseDouble(loc);
//Gera logs para verificacao posterior
log.info("******** BEFORE "+before+"************");
```

```
12
13 double uniform = Math.random();
14
15 double expRandom = -Math.log(uniform)/lambda;
16
17 log.info("******** EXP "+expRandom+"*************");
18
19 expRandom = expRandom+before;
20
21 //insere o  valor na variavel para a proxima Thread
22 props.put("beforeTime", expRandom.toString());
23
24 log.info("******** NEXT "+expRandom+"*************");
25
26 return ((int) expRandom);
```

D

Requisições Cloud EC2

Esse Apêndice apresenta o algoritmo para realizar chamadas pela API EC2. O método para medir o tempo de instanciar uma VM é *instantiateTime*, que executa 100 chamadas por tipo de VM.

```
1
2  public AmazonEC2Client getEC2(String cloudMngHost,String credentialFile)
    {
3    AmazonEC2Client ec2 = null;
4    AWSCredentials credentials = null;
5    String endpoint = cloudMngHost;
6    Protocol protocol = Protocol.HTTP;
7    int timeout = 600 * 1000;
8    try {
9      File file = new File(credentialFile);
10     credentials = new PropertiesCredentials(file);
11   } catch (Exception e1) {
12     System.out.println(e1);
13   }
14
15   ClientConfiguration client = new ClientConfiguration();
16   client.setProtocol(protocol);
17   client.setSocketTimeout(timeout);
18   ec2 = new AmazonEC2Client(credentials, client);
19   ec2.setEndpoint(endpoint);
20
21   return ec2;
22 }
23
24 public List<Instance> listInstances(AmazonEC2Client ec2) {
25   DescribeInstancesResult describeInstancesResult = ec2.describeInstances
    ();
26   List<Reservation> reservations = describeInstancesResult.
    getReservations();
27   List<Instance> allInstances = new ArrayList<Instance>();
```

```
28      for (Reservation reservation : reservations) {
29        List<Instance> instances = reservation.getInstances();
30        allInstances.addAll(instances);
31      }
32      return allInstances;
33    }
34
35    public List<Instance> searchInstances(String template, String state,
        String ip,AmazonEC2Client ec2) {
36      List<Instance> instances = new ArrayList<Instance>();
37      List<Instance> allInstances = listInstances(ec2);
38
39      for (Instance instance : allInstances) {
40        if (ip == null && instance.getImageId().equals(template) && instance.
        getState().getName().contains(state)) {
41          instances.add(instance);
42        } else if (instance.getImageId().equals(template) && instance.
        getState().getName().contains(state)
43            && instance.getPrivateIpAddress().contains(ip)) {
44          instances.add(instance);
45        }
46      }
47      return instances;
48    }
49
50    public void runConversor(String templateID,String type, AmazonEC2Client
        ec2) {
51      RunInstancesRequest runInstancesRequest = new RunInstancesRequest();
52      runInstancesRequest.withImageId(templateID).withInstanceType(type)
53          .withMinCount(1).withMaxCount(1)
54          .withSecurityGroups("default");
55      ec2.runInstances(runInstancesRequest);
56    }
57
58    public void deleteInstance(Instance instance,AmazonEC2Client ec2) {
59      TerminateInstancesRequest terminateInstancesRequest = new
        TerminateInstancesRequest();
60      terminateInstancesRequest.withInstanceIds(instance.getInstanceId());
61      ec2.terminateInstances(terminateInstancesRequest);
62
63    }
64
65    public void instantiateTime(String templateID,String cloudMngHost,String
        credentialFile) throws InterruptedException, InvalidKeyException,
        NoSuchAlgorithmException,
66    SignatureException, UnsupportedEncodingException {
67
```

```
68      ArrayList<String> result = new ArrayList<>();
69
70      Thread t;
71      Listner listner = new Listner();
72      //
73      t = new Thread(listner);
74      t.start();
75      AmazonEC2Client ec2 = getEC2(cloudMngHost,credentialFile);
76      ArrayList<String> types = new ArrayList<String>();
77      types.add("t2.micro");
78      types.add("t2.small");
79      types.add("t2.medium");
80      types.add("t2.large");
81      for (String type : types) {
82
83        for (int i = 0; i < 100; i++) {
84          result.add("criar instancia "+type+" " + i + " , " + (new Date()).
    getTime());
85          runConversor(templateID,type,ec2);
86          listner.accept();
87          result.add("receber instancia "+type+" " + i + " , " + (new Date())
    .getTime());
88          Instance instance = searchInstances(templateID, "run", null,ec2).
    get(0);
89          result.add("remover instancia "+type+" " + i + " , " + (new Date())
    .getTime());
90          deleteInstance(instance,ec2);
91          Thread.sleep(1000 * 30 * 1);
92        }
93        Thread.sleep(1000 * 60 * 5);
94      }
95
96    }
```