CHAVE: Consolidation with High Availability on Virtualized Environments

Daniel Scheidemantel Camargo

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA – DCC)
Universidade do Estado de Santa Catarina (UDESC) – Joinville/SC – Brasil
daniel@colmeia.udesc.br

Resumo. *Organizações e desenvolvedores cada vez mais demandam alta disponibilidade (HA) para serviços críticos. Contudo, nem sempre a taxa de disponibilidade estabelecida em Acordo de Nível de Serviço (SLA) supre as demandas de HA ou é devidamente cumprida. Plataformas de computação em nuvem Infraestrutura como Serviço (IaaS) nativamente fornecem mecanismos de HA para aplicações multicamadas, habitualmente baseadas em balanceadores de carga para serviços stateless, e por mecanismos de replicação em serviços auto-gerenciados, como em plataformas de banco de dados e armazenamento de objetos. Para todos os demais serviços a serem implementados em máquinas virtuais (MVs), há apenas a taxa de disponibilidade estabelecida em SLA. Assim, o presente trabalho apresenta um mecanismo de HA baseado em replicação passiva, transparente para o desenvolvedor que deve apenas indicar quais são as MVs críticas e taxa de alta disponibilidade requisitada. Todavia, todo e qualquer mecanismo de HA usualmente implica em mais equipamentos computacionais para MVs réplicas, que por conseguinte, aumentam o consumo de energia elétrica. Deste modo, estratégias de consolidação de MVs destacam-se como uma abordagem amplamente utilizada para reduzir o consumo de energia, alocando todas as MVs no menor número possível de servidores e permitindo desativar servidores ociosos. Porém, mecanismos de HA possuem a restrição de justaposição, em que as MVs primárias não devem estar posicionadas com as suas réplicas para não constituir um ponto único de falhas (SPoF). Esta restrição de justaposição deve orientar a estratégia de consolidação, para que não seja infringida nas etapas de posicionamento de novas requisições ou de migração das MVs. Uma solução adotada é a utilização de múltiplas zonas de disponibilidade (AZs), que são posicionadas as instâncias primárias e réplicas em AZs diferentes, fornecendo para a estratégia de consolidação apenas uma visão local, internamente a cada AZ. Deste modo, o presente trabalho realiza a simulação de nuvem IaaS no estudo de caso, variando entre as etapas de consolidação (migração e posicionamento) e algoritmos modificados baseados na heurística First Fit Decreasing (FFD), visando obter o menor número de migrações e reduzir o consumo de energia. São utilizados tanto cargas com valores aleatórios uniformemente distribuídos, quando cargas de traces Google Cluster e de nuvens privadas Eucalyptus. Como resultado, observa-se que devem ser sempre analisadas todas as estratégias, pois utilizando a sua combinação pode-se obter resultados diferenciados conforme o conjunto de requisições.*

1. Introdução

A computação em nuvem tem se tornado uma base fundamental para organizações que dependem da sua continuidade nos negócios baseada em serviços críticos, uma vez que a interrupção destes serviços pode causar prejuízos financeiros e de reputação [Maciel 2016, Alkawsi et al. 2015]. Neste sentido, o paradigma de computação em nuvem destaca-se por fornecer recursos computacionais sob demanda no modelo *pay-as-you-go*, com garantias de disponibilidade [Mell e Grance 2011] estabelecidas em Acordo de Nível de Serviço (SLA) entre o Provedor de Serviços em Nuvem (CSP) e o desenvolvedor (*tenant*, inquilino, ou entidade responsável pela implementação dos serviços). Entretanto, ainda é comum a interrupção de serviços por falhas do CSP, que vem ganhando cada vez mais atenção de pesquisadores [Islam e Manivannan 2017, Santos et al. 2017], transformando-se em um indicador para seleção de provedores para os desenvolvedores [Marston et al. 2011, Amoon 2016]. Estima-se que o impacto financeiro destas interrupções alcance a cifra de centenas de milhares de dólares por hora [Endo et al. 2016], não apenas para os desenvolvedores, mas também para CSPs.

A interrupção dos serviços pode ser causada internamente à aplicação, por erros de responsabilidade do desenvolvedor (*e.g.*, incorreta implementação/configuração dos serviços) [Beyer et al. 2016], ou a nível de virtualização e infraestrutura física, que são de responsabilidade do CSP [Endo et al. 2016]. Para reduzir ou mascarar as interrupções de responsabilidade do CSP, mecanismos de alta disponibilidade (HA) podem basear-se em métodos de redundância [Matos et al. 2017], replicação [Petrovic e Schiper 2012] e tolerância à falhas [Kanso et al. 2017], que podem ser implementados a nível de software ou de hardware [Prathiba e Sowvarnica 2017]. Todavia, todos estes mecanismos implicam em uma sobrecarga na infraestrutura física, pois demandam por mais equipamentos e elevam o custo total de propriedade (TCO), decorrendo em um maior consumo de energia elétrica. A energia é um recurso crítico e corresponde a até 50% dos custos operacionais em um Data Center (DC) [Comerford 2015].

Deste modo, o presente trabalho denominado CHAVE (*VM Consolidation with High Availability on Virtualized Environments*) tem o objetivo de mascarar as falhas na perspectiva do CSP, agindo especificamente na camada de processamento da nuvem (do servidor ao hipervisor) ao mesmo tempo em que realiza a consolidação de máquinas virtuais (MVs) que considera as restrições da HA. Esta restrição é denominada de justaposição de MVs, definida como "*MVs críticas e suas réplicas não devem ser instanciadas no mesmo servidor, por constituir o mesmo ponto único de falhas (SPoF)*" [Machida et al. 2010, Mondal et al. 2016]. Adicionalmente, o presente trabalho baseia-se na concepção de HA sob demanda, em que o desenvolvedor fornece quais são suas MVs críticas e qual a taxa de HA desejada para cada uma destas instâncias. Assim, CHAVE obtém a quantidade mínima de réplicas suficiente para fornecer a taxa de HA solicitada, reduzindo a probabilidade de falhas com um mecanismo de HA baseado em replicação por *checkpoint* no modo ativo-passivo. A atuação do mecanismo de HA na camada de processamento da nuvem permite uma abordagem agnóstica à aplicação, *i.e.*, independente dos serviços executados internamente à MV crítica, abstraindo-a como uma caixa-preta.

Para reduzir o impacto do consumo de energia em infraestruturas de computação em nuvem, uma das principais estratégias encontradas na literatura é a consolidação de MVs [Beloglazov e Buyya 2012, Ahmad 2015, Hameed et al. 2016]. Esta estratégia

visa alocar todas as MVs no menor número de servidores, possibilitando que equipamentos ociosos ou subutilizados sejam desativados/hibernados, além de permitir uma melhor alocação dos recursos virtuais no DC [Muchalski e Maziero 2014]. A consolidação de MVs é considerada uma aplicação prática do Problema do Bin Packing (PBP), de complexidade NP-Difícil [Martello e Toth 1990]. Entretanto, heurísticas permitem aplicar uma abordagem de consolidação mais rígida, *i.e.*, o mais próximo da configuração ótima, que colateralmente podem implicar em uma série de violações de SLA por infringir requisitos de disponibilidade [Nanduri et al. 2014,Hameed et al. 2016]. Assim, um mecanismo de consolidação deve ser desenvolvido consciente das restrições e propósitos do sistema em que é aplicado. Ao aplicar a consolidação de MVs simultaneamente a um mecanismo de HA, deve-se considerar a possibilidade de alocar uma MV crítica com suas réplicas em um mesmo SPoF afetando a HA desejada. Neste sentido, o presente trabalho aborda a restrição de justaposição da HA é solucionada para infraestruturas de DCs arquitetadas em múltiplas zonas de disponibilidade (AZs) [Unuvar et al. 2014, David 2014, Burns 2017,Moreno-Vozmediano et al. 2017], que consiste em afirmar que duas MVs replicadas não devem estar posicionadas na mesma AZ.

A contribuição do presente trabalho é fornecer um mecanismo de HA para MVs sob demanda para o desenvolvedor, vinculado a uma estratégia de consolidação de MVs que possibilita reduzir o impacto no uso de recursos, como a energia elétrica. Isto é, enquanto o mecanismo de HA permite prover serviços contínuos ao desenvolvedor, a consolidação permite reduzir o consumo de energia adicionado pelo mecanismo de HA. Adicionalmente, formaliza-se a restrição de justaposição de réplicas imposta pela HA com a definição uma Programação Linear Inteira (PLI) para a consolidação de MVs a nível de arquiteturas Multi-AZ. A abordagem de HA sob demanda, permite ao desenvolvedor selecionar o nível de HA que deseja para cada uma de suas instâncias críticas. Porém, para instâncias regulares que não demandam HA (não consideradas críticas), a taxa regular especificada em SLA deve suprir a demanda. O presente trabalho está organizado da seguinte forma. A Seção 2 aborda a motivação e fundamentação teórica utilizada no decorrer do artigo, na Seção 3 é feito um levantamento e discussão dos trabalhos correlatos. A proposta do presente trabalho é desenvolvida na Seção 4, que tem seu plano de testes organizados na Seção 5 com a discussão dos resultados na Seção 6. Por fim, as considerações finais estão descritas na Seção 7 Por fim, são analisados e discutidos os resultados da aplicação da solução proposta em uma simulação em plataforma nuvens computacionais baseadas em Infraestrutura como Serviço (IaaS).

2. Motivação e formulação do problema

A disponibilidade é um requisito amplamente utilizado para indicar o tempo de execução de serviços ou sistemas em um determinado período de tempo [Wu e Buyya 2015]. Por definição, a disponibilidade refere-se a probabilidade de um serviço ou sistema estar operando corretamente quando requisitado para uso [Stapelberg 2009, Bauer e Adams 2012, Critchley 2014]. Genericamente, a disponibilidade é especificada pela razão entre o tempo de atividade e o tempo total observado. Em geral, interrupções podem ser causadas por dois principais motivos: manutenção (preventiva ou preditiva) e ocorrência de falhas aleatórias (não programadas), sendo que para sistemas distribuídos, como as plataformas de computação em nuvem, a manutenção pode ser realizada de forma planejada [Riasetiawan et al. 2015]. Neste sentido, para sistemas computacionais distribuídos

a literatura especializada [Weibull 2007, Stapelberg 2009, Bauer e Adams 2012] baseia-se na disponibilidade inerente, que relaciona apenas o tempo de atividade do sistema e o tempo de manutenção corretiva, desconsiderando atrasos logísticos e manutenção preventiva/preditiva. O tempo de manutenção corretiva (T_{mc}) refere-se ao período necessário para recuperar totalmente um serviço ou sistema desde a detecção de sua falha até a comutação para o componente substituto. De forma genérica, para obter a disponibilidade inerente $\mathbb{A}$ para um único componente, aplica-se a Equação 1 em um período determinado de tempo t, sendo $T = t_{final} - t_{inicial}$.

$$\mathbb{A} = \frac{T_a}{T_{to}} = \frac{T_a}{T_a + \sum_{i=0}^{\mathrm{n}} T_{mc\ i}} \tag{1}$$

Na Equação 1, o tempo de atividade (T_a) e o tempo total de manutenção corretiva ($T_{to} = \sum T_{mc_i}$) para n falhas em um período t podem ser obtidos por análise histórica [Zhou et al. 2017, Moreno-Vozmediano et al. 2017] ou por indicadores probabilísticos [Koslovski et al. 2010]. Fatores históricos consistem em análise de séries temporais de atividades e falhas ocorridas no passado, relacionando períodos de atividades com períodos de manutenção. Indicadores probabilísticos são taxas de confiabilidade atribuídas a determinados equipamentos de hardware pelos fabricantes nos *datasheets*, destacando-se o tempo médio entre falhas (MTBF) para sistemas reparáveis, o tempo médio para falhar (MTTF) para sistemas não reparáveis, além do tempo médio para reparo (MTTR), relativo ao tempo de manutenção corretiva. Adicionalmente, observa-se na Equação 1 que a taxa de disponibilidade $\mathbb{A}$ tende a ser maior a medida em que o número de falhas (n) e o tempo de manutenção corretiva (T_{mc}) são minimizados. Como é inviável minimizar o número de falhas, pois elas são aleatórias e imprevisíveis, utilizar mecanismos de alta disponibilidade (HA) permitem minimizar o tempo de manutenção corretiva do sistema, em procedimento denominado de tolerância a falhas [Mondal et al. 2016] que é considerado como a comutação da máquina virtual (MV) falhada para sua réplica.

No contexto do presente trabalho, os servidores de uma determinada zona de disponibilidade (AZ) são os componentes com maior probabilidade de falhas, e cada AZ tem uma menor probabilidade de falhas, representando o mais alto nível de ponto único de falhas (SPoF). Define-se AZs como *clusters* de equipamentos isolados ou Data Centers (DCs) separados das demais AZs, que executam uma plataforma de nuvem distribuída em conjunto. Essa isolamento possibilita que outras AZs não serão afetadas na ocorrência de falhas diversas na infraestrutura da AZ, para que continuem operacionais. Esta característica das AZs habilitou uma nova arquitetura de serviços *stateless* nativos para a nuvem [Balalaie et al. 2016], multicamadas e sem dependência de estados transientes de memória e registradores, que se beneficiam da HA com o uso de balanceadores de carga em MVs efêmeras. Todavia, serviços *stateful* baseados em estados transientes, ainda demandam uma replicação explícita dos estados de memória e dos registradores. São exemplos de sistemas *stateful* as operações de banco de dados, aplicações de Computação de Alto Desempenho (HPC) e sistemas legados que ainda não foram devidamente portados para a nuvem. Embora cada vez mais novos sistemas estejam sendo desenvolvidos nativamente para a nuvem [Villamizar et al. 2015], ainda há uma diversidade de sistemas que requisitam a replicação explícita dos estados de memória dos seus serviços.

Segundo Critchley 2014, define-se alta disponibilidade (HA) como a capacidade

de um sistema fornecer serviços quando requisitado, em qualquer ponto no tempo e durante um período de tempo determinado. A literatura especifica uma taxa acima de cinco noves ($99,999\%$) como classificação para um serviço sob HA [Endo et al. 2016, Moreno-Vozmediano et al. 2017]. No contexto de computação em nuvem, a replicação de MV é uma abordagem que possibilita obter maiores taxas de HA [Mondal et al. 2016]. A replicação consiste em distribuir n MVs em SPoF diferentes [Fischer e Mitasch 2006], atualizando as MVs réplicas com os estados da MV primária, no modo passivo, ou executando todas as n MVs em paralelo no modo ativo [Endo et al. 2016]. No modo ativo, normalmente a consistência dos estados de memória e CPU das MVs podem ser periodicamente comparados, em uma abordagem denominada *lock step* [Dong et al. 2013]. Naturalmente, a replicação ativa demanda uma maior carga de trabalho (processamento e rede) e maior complexidade no uso de recursos, quando comparada com a replicação passiva [Fischer e Mitasch 2006, Zhou et al. 2017]. Assim, o presente trabalho baseia-se apenas na replicação passiva, devido a sua baixa complexidade de implementação e por ser amplamente usada em diferentes propostas [Helal et al. 2006, Endo et al. 2016]. A replicação passiva remete a configuração para a tolerância a falhas no nível *Warm Standby* [Matos et al. 2017], que geralmente apresenta um tempo de recuperação de poucos segundos.

A análise por diagramas de blocos de confiabilidade (RBD) é amplamente adotada para calcular a disponibilidade de sistemas complexos [Stapelberg 2009, Santos et al. 2017, Matos et al. 2017], verificando se os componentes estão em série ou em paralelo. Quando componentes estão em série, *i.e.*, no mesmo SPoF, a disponibilidade combinada é o produto da disponibilidade '$\mathbb{A}$' de seus k componentes, que sempre resulta em uma disponibilidade menor do que a de seus componentes individuais e, portanto, não aplicado para HA. Para componentes em paralelo, a disponibilidade combinada é baseada no produto da indisponibilidade $(1 - \mathbb{A})$ de seus k componentes [Coulouris et al. 2011]. Considera-se, portanto, que quando os k componentes não compartilham o mesmo SPoFs, então não haverá dependência entre as falhas, *i.e.*, a ocorrência de uma falha no componente ativo não será a responsável pela falha no componente replicado, e portanto, se reduz a probabilidade de todos os componentes falharem [Fischer e Mitasch 2006, Ford et al. 2010, Moreno-Vozmediano et al. 2017]. Este é um axioma proveniente da probabilidade de eventos independentes, expresso pela Equação 2. Assim, quando em paralelo, a disponibilidade combinada é sempre maior do que a disponibilidade média dos seus componentes individuais. A Equação 2 permite calcular a taxa de HA para k componentes em paralelo, que é a soma do componente crítico e as suas r réplicas. Em sistema mais complexos, com diversos componentes em série e em paralelo, deve-se calcular a disponibilidade utilizando análise de RBD por partes [Matos et al. 2017].

$$\mathbb{HA} = 1 - (1 - \mathbb{A})^k \mid k = \text{Num. de Paralelos} \tag{2}$$

Mecanismos de HA baseados em replicação de MV fornecem sincronização de estados de memória e processamento, garantindo que não haja a perda de serviço ou dados durante a recuperação. A abordagem denominada de *checkpoint* é essencial para serviços que mantém seus estados em memória (*stateful*), característico de sistemas legados, não portados adequadamente como uma aplicação nativa para nuvem [Bias 2016]. Para a detecção de falhas a nível de MVs, deve-se contar com mecanismos que verifi-

quem continuamente a condição dos componentes ativos, habitualmente em abordagens por *heartbeats* ou protocolo *gossip*. Estas abordagens devem ter a habilidade de detectar rapidamente a falha para recuperar automaticamente migrando a conexão do serviço para o componente redundante [Yang et al. 2011]. A camada de controle de uma nuvem computacional normalmente fornece um nível adequado de detecção, isolamento e recuperação de falhas, habitualmente na camada de armazenamento de dados através de sistemas de arquivos distribuídos [Zhang et al. 2015], e na própria camada de controle com o uso de *clusters* [Wang e Li 2015]. Assim, se uma falha for detectada pelo mecanismo de HA, a MV poderá ser reiniciada em um servidor diferente, resultando em perdas de informações em memória que geralmente não são preservadas para serviços *stateful* [Bauer e Adams 2012]. Todavia, tanto na replicação quanto em qualquer outro mecanismo de HA, há uma necessidade inerente de mais servidores de processamento e mais conexões de rede, que acrescentam o custo total de propriedade (TCO) e uma demanda de maior consumo de energia, que é uma das principais despesas de um Provedor de Serviços em Nuvem (CSP).

A estratégia de consolidação de MVs é aplicada no presente trabalho para reduzir o impacto do mecanismo de HA, como a quantidade de servidores ativados possibilitando reduzir o consumo de energia. Como o ambiente de nuvem é dinâmico e sua carga varia com o tempo, a consolidação baseia-se em dois mecanismos que ocorrem em momentos distintos: o posicionamento inicial e a migração de MVs [Abdelsamea et al. 2017]. O posicionamento inicial consiste em selecionar os servidores mais adequados para instanciar as novas requisições para criação de MVs. A migração ocorre periodicamente, e consiste em mover uma MV em execução para outro servidor, visando aprimorar o objetivo geral da consolidação. Na prática, quando executa-se a migração de uma MV, há a transferência de um conjunto de processos, com suas páginas de memória e estados do processador, entre os servidores de origem e de destino, reconfigurando as conexões de rede na camada de controle da nuvem.

A consolidação de MVs é uma exemplo prático do Problema do Bin Packing (PBP), de complexidade NP-Difícil. Como solução para esta complexidade, a literatura propõe diversas heurísticas para obter uma aproximação do valor ótimo (OPT) [Martello e Toth 1990, Zhang et al. 2017, Rieck 2010], que podem variar desde abordagens simples como o *Best-Fit* até meta-heurísticas, como as bioinspiradas em colônias de formigas [Ferdaus et al. 2014]. Entre estas abordagens, deve-se destacar a necessidade de selecionar uma heurística compatível com as características do problema em que será aplicado, observando a relação custo-benefício de obter a resposta mais aproximada do valor ótimo em um tempo aceitável. É inerente que para todas as aplicações exige-se uma resposta mais aproximada da ótima, pois o PBP exige uma solução de otimização. Porém, ao considerar que o tempo de *boot* entre solicitar a alocação de uma MV e conectá-la ao usuário está na ordem de minutos (dois minutos em média para a AWS) [Nguyen e Lebre 2017, Kominos et al. 2017], para o contexto do presente trabalho é aceitável que poucos segundos de execução do algoritmo heurístico não cause impactos no tempo de *boot* total. Em uma análise realizada no *benchmark* específico para o PBP [Rieck 2010] com seis principais heurísticas: *Max-Rest, Next-Fit, Next-Fit-Decreasing, Best-Fit, First-Fit* e o *First-Fit-Decreasing* indicam uma grande variação nos critérios de aproximação do valor ótimo e o tempo de execução para um determinado conjunto de itens. Assim, o *First Fit Decreasing* (FFD) destaca-se por apresentar os valores mais aproximados do ótimo,

embora seu tempo de execução demande 0,2 segundos para consolidar 2,5 milhões de itens (o dobro do algoritmo mais rápido).

Diversas técnicas de implementação podem ser aplicadas no algoritmo FFD, na parte de ordenação inicial, na estrutura de dados, e em abordagens adicionais relacionadas ao problema em que é aplicado [Alahmadi et al. 2014]. Na análise de otimalidade, o FFD apresenta no pior caso um comportamento denotado por $FFD(L) \leq (11/9) * OPT(L) + n$ [Beloglazov e Buyya 2012, Rieck 2010, Dósa et al. 2013], sendo que n pode variar conforme as técnicas de implementação. Por exemplo, dado um valor ótimo $OPT(L) = 9$, então o FFD pode fornecer, no máximo, um $FFD(L) = 11 + n$.

Devido sua necessidade de ordenar o vetor de MVs antes de instanciá-los nos servidores, o algoritmo FFD é caracterizado como um algoritmo *offline*. Porém, como o FFD possui uma visão global para a sua tomada de decisões, permite-se obter melhores resultados quando comparado com uma abordagem de perspectiva mais restrita, como ocorre com os algoritmos *online*. Observando o presente cenário de aplicação do FFD: plataformas de nuvem Infraestrutura como Serviço (IaaS), observa-se a necessidade de um algoritmo *online* para atender imediatamente cada uma das requisições. Todavia, deve-se observar que quando há diversas requisições concorrentes, *i.e.*, que ocorrem no mesmo instante de tempo, obrigatoriamente será criada uma fila de requisições. Assim, o presente trabalho possui uma abordagem híbrida, em que o FFD tem como entrada esta fila instantânea de requisições, executada continuamente conforme os lotes de entradas *online*. Esta abordagem híbrida permite unir os benefícios de um algoritmo *online* que é atender *quasi*-instantaneamente as requisições, e *offline* que é encontrar os melhores resultados. Deste modo, o FFD deve ter como entrada uma lista $\mathcal{V}$ com n requisições e considerar a ocupação do servidores atualmente em execução.

De um modo geral, o procedimento de consolidação de MVs, possui o objetivo de alocar todas as MVs no menor número possível de servidores e desativar os servidores ociosos. Para isso, são definidas duas etapas, sendo uma relacionada ao posicionamento inicial das requisições, e outra que deve realizar a migração das MVs já instanciadas. Porém, conforme a configuração atual dos servidores em execução, e das características do conjunto de requisições a serem instanciadas, pode-se obter resultados distintos, que ora migrar antes de posicionar fornece um melhor resultado, ora posicionar antes de migrar é melhor. Assim, a presente proposta analisa ambas as possibilidades para auxiliar na tomada de decisão, resultando em uma abordagem agnóstica à estratégia.

Observa-se que pode ser conflitante realizar a consolidação de MVs concomitante a um mecanismo de HA, devido a inerente restrição de justaposição de MVs primárias e suas réplicas no mesmo SPoF. Ainda que o escalonador da plataforma de nuvem considere evitar a justaposição ao alocar as réplicas, infraestruturas que usam estratégias de consolidação de MVs podem ficar impedidas de obter resultados mais aproximados da configuração ótima. Isto ocorre pelo fato estar sendo adicionada mais uma restrição além da própria restrição da consolidação, que é verificar se uma MV pode ser instanciada em um servidor pela análise dos seus recursos físicos remanescentes. Observa-se também que as AZs de uma determinada plataforma de nuvem visam eliminar o SPoF. Na prática, plataformas de nuvem públicas como a *Amazon Web Services* (AWS) e *Google Cloud Platform* (GCP) possuem, uma média de 2,75 a 3 AZs por região geograficamente dis-

tribuída, sendo que algumas regiões contam com mais de três AZs[1]. Assim, também para outros modelos de implantação, como as nuvens privadas, que desejam prover serviços críticos através de mecanismos de HA, devem dispor de um número adequado de AZs para garantir as taxas de HA requisitadas. Assim, o presente trabalho surge com a abordagem de replicação Multi-AZ que habilita associar uma estratégia de consolidação de MVs para mitigar o impacto energético e de recursos causados pelo mecanismo de HA.

3. Trabalhos correlatos

A literatura apresenta duas linhas de pesquisa com objetivos distintos: a consolidação de máquinas virtuais (MVs) e mecanismos de alta disponibilidade (HA), mas que podem se complementar quando o objetivo é prover HA com baixo impacto no consumo de recursos. Em análise exploratória não-exaustiva, observam-se poucos trabalhos correlatos que abordam simultaneamente a consolidação e HA. Assim, são relacionados trabalhos referentes apenas à consolidação de MVs, apenas à mecanismos de HA e, por fim, trabalhos que abordam simultaneamente as duas linhas de pesquisa. Como trabalhos relacionados, buscou-se, preferencialmente, trabalhos com escopo e aplicação em computação em nuvem Infraestrutura como Serviço (IaaS).

3.1. Consolidação de MVs

Existem diferentes estratégias que objetivam a redução de consumo de energia em ambientes de computação em nuvem, destacando-se o escalonamento de tarefas, balanceamento de carga, alocação de recursos, *Dynamic Voltage and Frequency Scaling* (DVFS) e a consolidação de MVs. Segundo a literatura especializada, a estratégia da consolidação de MVs possui um dos resultados mais significativos [Beloglazov e Buyya 2012, Medina e García 2014, Ahmad 2015, Madni et al. 2016, Milani e Navimipour 2016, Patel e Vaghela 2016, Hameed et al. 2016, Xu et al. 2016], pois podem alcançar até 40% de redução no consumo de energia. Todavia, dependendo da abordagem, efeitos negativos podem ocorrer com as violações de Acordo de Nível de Serviço (SLA) devido ao alto número de migrações necessário para atingir resultados que tendem ao ótimo.

Um dos primeiros trabalhos a abordar a consolidação para o gerenciamento de energia no contexto de virtualização em data centers é de Nathuji e Schwan 2007. Os autores propõem uma arquitetura de gerenciamento de recursos organizados em políticas locais, a nível de MV, e globais, para *racks* e Data Center (DC). A nível local, a solução prioriza estratégias de gerenciamento de MVs através do acesso às características do serviço/aplicação executado na MV do desenvolvedor, para selecionar políticas específicas que permitam otimizar o gerenciamento. A nível global, a solução obtém as informações sobre a alocação de recursos atuais, além das características de nível local, e aplica as políticas para decidir se o posicionamento da MV precisa ser adaptado através de migração. Todavia, a solução não mostra nenhuma política específica que automatize o gerenciamento de recursos a nível global. Acessar o espaço do desenvolvedor e coletar características sobre a carga e natureza da aplicação, viola preceitos de segurança. Assim, a presente proposta visa realizar a consolidação considerando apenas os recursos efetivamente alocados, que na prática são as informações básicas para toda estratégia de

[1]Fonte da própria organização AWS: `https://aws.amazon.com/pt/about-aws/global-infrastructure` e GCP `https://cloud.google.com/compute/docs/regions-zones`.

consolidação. A nível global, a presente proposta baseia-se na organização de conjuntos de servidores em diversas zonas de disponibilidade (AZs), que por ser previamente configurada pelo desenvolvedor, não podem haver migrações para outras AZs .

O trabalho de Corradi et al. 2012 realiza a implementação prática de consolidação de MVs no OpenStack[2], utilizando medições de consumo com wattímetro em servidores commodities. Seu objetivo é avaliar os parâmetros de uso de CPU e rede em um nível máximo de consolidação que propositalmente resulte em diversas violações do SLA. É utilizada uma suíte de testes com migração de MVs para propositalmente resultar na queda no desempenho das aplicações executadas nestas MVs. Os autores baseiam-se apenas na migração de MVs para maximizar o uso e alocação de recursos físicos, mas sem considerar qualquer estratégia em especial, pois os servidores e MVs são selecionadas manualmente. Assim, os autores concluem que, em teoria, a consolidação não mostra seus potenciais efeitos, e que na prática deve-se considerar o desempenho das aplicações. CHAVE busca minimizar o número de migração de MVs, visando utilizar o posicionamento inicial aliada a etapa de migração apenas o suficiente para desativar servidor subutilizados.

O trabalho de Beloglazov 2013 utiliza as técnicas de migração e posicionamento para realizar a consolidação, aplicadas tanto por simulação quanto em um ambiente real com o OpenStack. O autor aplica a heurística *First Fit Decreasing* (FFD) para obter um resultado mais próximo do ótimo, considera as violações do SLA como métrica (SLAV), mas observa apenas a sobrecarga e a degradação do desempenho devido as migrações. Todavia, como seu objetivo é de buscar o nível ótimo de consolidação, sua estratégia não considera outras restrições como a co-localização/justaposição de MVs no mesmo servidor, o que na prática é um requisito a ser consideradas. CHAVE considera desde sua formalização, a existência de restrições que naturalmente coexistem em um ambiente de nuvem real, harmonizando-se com mecanismos de HA, e mostrando que é possível alinhar uma estratégia de consolidação com demais restrições impostas por outros problemas.

Zhang et al. 2014 realiza a implementação de um escalonador consciente de energia em uma nuvem privada Eucalyptus, com uma abordagem apenas no posicionamento inicial. Os autores implementaram três políticas de escalonamento para comparar com as políticas nativas no Eucalyptus, utilizando uma correlação entre recursos de físicos e virtuais. Assim, a consolidação ocorre quando os servidores ociosos são desativados ou postos em hibernação, a medida em que são finalizadas as instâncias anteriores. A sua abordagem não utiliza as migrações de MVs para o processo de consolidação, o que demora a convergir para uma configuração de consolidação mais próxima do ótimo. CHAVE utiliza as duas estratégias: posicionamento inicial novas instâncias nos servidores ativos conforme seus recursos remanescentes, e a migração das MVs para consolidar servidores subutilizados.

Todos os trabalhos relacionados tem a consolidação de MVs como único objetivo de sua proposta, não considerando outras restrições reais que o ambiente de nuvem possa ter, como em segurança no que refere-se ao acesso às características de carga de trabalho e ao espaço de aplicação do desenvolvedor), e disponibilidade, com as violações de SLA. O presente trabalho aborda a consolidação sob um ponto de vista prático, com atenção

[2]Solução de computação em nuvem open source, disponível em: `http://www.openstack.org`.

específica aos aspectos relacionados com a capacidade de harmonizá-la às restrições existentes, como as impostas pela HA.

3.2. Alta disponibilidade

A revisão de Ranjan et al. 2015 mostra que nuvens públicas como a *Amazon Web Services* (AWS), permitem que um desenvolvedor habilite a replicação Multi-AZ apenas em serviços gerenciados, como em banco de dados e outras aplicações fornecidas como *Plataform as a Service* (PaaS). Assim, já é uma prática oferecer HA como um recurso adicional, mas apenas a nível de aplicação. Para provedores IaaS, a replicação de MVs ainda não é um recurso oferecido, sendo o mais próximo disso serviços de *snapshots* contínuos com a migração destas imagens (AMI) entre AZs, sendo uma operação manual ou por API que deve ser realizada pelo desenvolvedor.

Masakari [Masakari 2017][3] é uma solução que atua na recuperação de falhas na camada de virtualização, atuando nos processos gerenciados pelo hipervisor KVM do OpenStack. Quando uma MV com serviços *stateless* falha, o Masakari reinicia os processos do hipervisor relacionados à esta MV no mesmo servidor ou em outro. Porém, para MVs com serviços *stateful* este procedimento causa a perda de todos os estados que não estavam persistidos (dados em estado transiente), tornando-se uma solução que não é aplicável em todos os tipos de serviços. A nível de instância, a abordagem consiste em aplicar um *stop-start* na MV, equivalente a uma reinicialização forçada. Para falhas a nível de servidor, aplica-se um procedimento de evacuação, que implica em reiniciar todas as MVs que estavam instanciadas no servidor falhado em outros servidores capazes de recebê-las. Para ambas as abordagens, o Masakari é uma solução recomendável apenas para serviços *stateless*, enquanto CHAVE possui uma abordagem de replicação que pode ser utilizada tanto para serviços *stateful* quanto *stateless*.

Kemari Tamura et al. 2008 é uma solução de replicação por *checkpoint*, que captura os estados de memória, rede, armazenamento e contexto do processador da MV ativa e transfere para a MV passiva. A ativação do *checkpoint* ocorre apenas quando eventos externos ocorrem, como a escrita em disco ou envio de pacotes para a rede. Inicialmente foi desenvolvida para hipervisores Xen e posteriormente suportada para QEMU e KVM. Entretanto, Kemari não possui detecção de tolerância a falhas automática, como o Corosync e Pacemaker, utilizados em outras soluções de HA. A tolerância a falhas é um mecanismo importante para realizar a troca de atribuição ativo-passivo no instante de tempo em que ocorre a falha. Adicionalmente, a solução permite apenas a replicação de 1:1, não é documentada a possibilidade de ter mais de uma réplica para cada MV ativa. A abordagem deste trabalho correlato tem uma proximidade com o ideal, para ser considerado para a solução proposta, por basear-se em uma solução de replicação por *checkpoint* para os hipervisores mais utilizados em soluções de nuvem *open source*. Embora careça de uma abordagem de *multicast* para prover replicação de 1:N e tolerância a falhas automáticas.

Assim como o Kemari, Remus [Cully et al. 2008] realiza a replicação através de *checkpoint*, porém em intervalos regulares de tempo. Uma característica importante do Remus é que, em cada *checkpoint*, a saída externa é armazenada em um *buffer* local no servidor crítico até que seja assegurado que o servidor de *backup* conclua essa atualização

[3]Disponível em: `https://wiki.openstack.org/wiki/Masakari`.

do *checkpoint*. Isso garante que qualquer operação de tolerância a falhas seja transparente para outros servidores não afetados.

Além disso, o servidor crítico continua a sua execução em paralelo até o próximo *checkpoint*, aumentando assim o desempenho do sistema em relação ao controle clássico por operações bloqueadas. A partir da versão 4.0.0, o Remus está incluído nas versões oficiais do Xen-Project[4]. Considera-se o Remus como uma solução ideal para ser adotada como mecanismo de HA para o presente trabalho.

COLO [Dong et al. 2013] é uma solução de replicação que se utiliza do conceito de *lockstep*, que compara a saída dos pacotes de rede para verificar se ambas as MVs crítica e réplica estão consistentes entre si, realizando uma abordagem de *checkpoint* periodicamente.. Testes indicam uma redução de carga de CPU e memória de até 80% em relação ao desempenho nativo de uma MV, sendo sua comparação feita com Remus em diferentes *benchmarks*. Porém, para permitir a comparação dos pacotes de saída e comparar as MVs críticas e réplica, COLO tem que utilizar uma versão modificada do protocolo TCP, sendo esta uma abordagem invasiva que mantém o desenvolvedor dependente de protocolos particulares internamente à sua MV crítica. Embora CHAVE não vise a funcionalidade de *lockstep*, também não necessita de alterações dos protocolos de rede e comunicação internamente à MV, embora COLO indica melhor desempenho do que as demais abordagens de replicação.

3.3. Consolidação de MVs com Alta disponibilidade

Para prover alta disponibilidade com MVs k-resilientes, Bin et al. 2011 propõe um algoritmo de alocação de recursos virtuais baseado no uso de MVs-sombra, implantadas em máquinas físicas específicas para fornecer serviços de *backup*. Uma estratégia de consolidação de MVs dinâmica ajusta o posicionamento de MVs para se adaptar à demanda variável de recursos. Em seu mecanismo de HA, utiliza-se replicação por *snapshots* em *Cold Standby*, uma abordagem que periodicamente realiza *backup* de toda a MV (Kernel, Sistema Operacional (SO), aplicações e estados de memória), persistido em um sistema de arquivos distribuído. Isso permite instanciá-la em qualquer outro servidor, mas o intervalo de tempo demandado para esse mecanismo cria uma interrupção que chega a dezenas de minutos, dependendo das características da instância. Adicionalmente, este trabalho aborda o problema de orientar o posicionamento com as restrições impostas pela HA para não constituir o mesmo ponto único de falhas (SPoF) entre MV crítica e *backup* (justaposição).

[Simonin et al. 2013] apresenta uma solução para plataforma de nuvem privada com foco em escalabilidade, eficiência energética e tolerância a falhas. A solução permite aos desenvolvedores implementarem infraestruturas virtuais e controlar o ciclo de vida das suas MVs. Porém, apenas o administrador da nuvem pode configurar o nível de tolerância a falhas que o gerenciador deve realizar a auto-cura (*self-healing*) como um mecanismo de HA, não deixando a critério do desenvolvedor qual o nível de HA ele deseja, portanto, não suporta HA sob demanda. A consolidação é feita apenas por um mecanismo de posicionamento, *i.e.*, não utiliza a migração de MVs, e gerencia a distribuição de MVs com foco no gerenciamento de energia, detectando a sub/superutilização de recursos físicos. A distribuição física dos servidores segue uma arquitetura *Big-Tree*, similar

[4]Disponível em: `https://www.xenproject.org/`.

a arquitetura Multi-AZs utilizada para providenciar a redundância apenas em equipamentos de rede e servidores de controle. São utilizados mecanismos de verificação de falhas (*health-check*), como *heart-beat*, *self-healing* e *autorecovery* para prover HA. Porém, no trabalho relacionado não são relatados testes sobre a consolidação de MVs e dados sobre eficiência energética, bem como informações sobre as características dos mecanismos de HA utilizados.

[Li et al. 2016] propõe uma solução de HA baseada em uma heurística de uso de MVs com tempo compartilhado, baseada no problema do *multi-armed bandit*, da teoria da probabilidade. Esta proposta objetiva reduzir a utilização dos recursos de memória e armazenamento das MVs de *backups* enquanto fornece HA através da replicação na camada de aplicação, interno ao espaço do desenvolvedor. Os experimentos com simulação indicam que a tradicional relação de 1:1 para de replicação pode estender-se para 1:M, *i.e.*, M serviços para cada *backup*. Uma das principais métricas utilizadas é o tempo de utilização de cada *backup* por serviço e as relações de utilização entre estes serviços de modo compartilhado. Porém, na prática é inviável para o Provedor de Serviços em Nuvem (CSP) consolidar diversos serviços em apenas uma MV, pois torna-se necessário ter acesso às características e aos serviços de cada desenvolvedor, o que viola preceitos de isolamento e segurança da informação. CHAVE objetiva uma aplicação prática, alinhada com os requisitos reais tanto de um CSP quando do desenvolvedor.

3.4. Considerações parciais sobre os trabalhos correlatos

A literatura especializada indica que há diversas abordagens para as problemáticas definidas na presente proposta, indicando que esta é uma área de potencial demanda, tanto para pesquisa quanto para os negócios. Porém, até o momento não foi encontrado uma abordagem que envolva todas as problemáticas em apenas uma solução. Assim, o presente trabalho apresenta uma proposta para fornecer uma solução de HA que, harmonizada a uma estratégia de consolidação de MVs, permite reduzir o impacto no uso de recursos físicos, permitindo que o desenvolvedor tenha acesso a HA sob demanda e como um serviço. A Tabela 1 apresenta uma relação entre os trabalhos correlatos, indicando as linhas de pesquisa de cada trabalho, suas contribuições e problemática não solucionadas.

Observando a Tabela 1 a coluna '*linha de pesquisa*' indica se o trabalho correlato aborda apenas a consolidação de MVs, apenas HA ou se une consolidação com HA. Percebe-se que há mais trabalhos correlatos com as linhas de pesquisa apenas em consolidação ou apenas em HA do que trabalhos que, de certo modo, relacionam as duas linhas de pesquisa ao mesmo tempo. As '*contribuições*' são as abordagens que possuem alguma relação com as propostas em CHAVE. As '*problemáticas*' consiste nas abordagens que não foram resolvidas por suas soluções, e que CHAVE agrega em sua proposta.

4. Solução proposta: CHAVE

O objetivo da solução proposta é fornecer um mecanismo de alta disponibilidade (HA) sob demanda aos desenvolvedores, reduzindo o impacto nos recursos físicos com uma estratégia de consolidação de máquinas virtuais (MVs). O mecanismo de HA atua no plano de processamento de uma nuvem computacional (*i.e.*, do servidor ao hipervisor), utilizando o modo de replicação passiva por *checkpoint*. Na concepção de HA sob demanda, o desenvolvedor solicita uma taxa de HA para cada uma das suas MVs críticas e CHAVE

Trabalhos correlatos	Linha de pesquisa	Contribuições	Problemáticas
Nathuji e Schwan 2007	Consolidação	Políticas locais e globais	Viola preceitos de isolamento e segurança
Corradi et al. 2012	Consolidação	Impacto da consolidação	Sem uma estratégia definida
Beloglazov 2013	Consolidação	Usa o FFD para simulação	Desconsidera restrições adicionais
Zhang et al. 2014	Consolidação	Correlaciona recursos físicos/virtuais	Utiliza apenas o posicionamento
Ranjan et al. 2015	HA	Replicação Multi-AZ por nuvens públicas	Trabalho é uma taxonomia
Masakari 2017	HA	Evacuação de MV e reinício forçado	Apenas serviços *stateless*
Tamura et al. 2008	HA	Replica por *checkpoint* e é compatível com diversos hipervisores	Sem tolerância a falhas automática e apenas replicação 1:1
Cully et al. 2008	HA	Possui *checkpoint* e replicação de 1:N tolerância a falhas automático	Compatível apenas com Xen
Dong et al. 2013	HA	Abordagem em *Lockstep* e provê melhor desempenho no uso de recursos	Invasivo, pois requer TCP da MV modificado
Bin et al. 2011	Consolidação com HA	Restrição de justaposição	Replica por *snapshots*, maior tempo entre *backups*
Simonin et al. 2013	Consolidação com HA	Arquitetura *Big-Tree*, verificação de falhas	Sem HA sob demanda do desenvolvedor
Li et al. 2016	Consolidação com HA	Replicação 1:M, conceitos probabilísticos	*Backup* por *snapshots*, viola preceitos de isolamento e segurança

Tabela 1. Relação dos trabalhos correlatos.

as replica para r MVs réplicas instanciadas em servidores localizados em diferentes zonas de disponibilidade (AZs) visando garantir a taxa desejada. O mecanismo de HA é harmonizado com uma estratégia de consolidação de MVs para reduzir seu inerente impacto na utilização de recursos físicos, mitigando o custo total de propriedade (TCO) e seu respectivo consumo de energia elétrica. Assim, é proposto um algoritmo de consolidação consciente das restrições de HA, como a justaposição ou co-localização de MVs críticas e réplicas no mesmo ponto único de falhas (SPoF).

CHAVE utiliza conceitos de diagramas de blocos de confiabilidade (RBD) para análise de disponibilidade, embasada na probabilidade de falhas independentes (Equação 2) para especificar o número de réplicas necessárias para prover a taxa de HA desejada. Quando isolado o número de paralelos k, conforme a Equação 3, obtêm-se o número de réplicas r necessárias para fornecer uma determinada taxa de $\mathbb{HA}$. Como o valor de k e r devem ser discretos, utiliza-se a função *ceil*, que corresponde ao menor valor inteiro maior ou igual a r, *i.e.*, arredondado para cima.

$$\text{ceil}(r) \approx log_{1-\mathbb{A}}(1-\mathbb{HA}) - 1 \equiv \frac{log_{10}(1-\mathbb{HA})}{log_{10}(1-\mathbb{A})} - 1 \tag{3}$$

Na relação entre k (Equação 2) e r (Equação 3), ocorre que k é o número de paralelos, que inclui o próprio componente primário, enquanto r é o número de réplicas, *i.e.*, $r = k - 1$. Embora CHAVE não considere uma estratégia de posicionamento das réplicas em servidores diferentes da mesma AZ, salienta-se que, neste caso, a taxa de disponibilidade base $\mathbb{A}$ deve ser a observada a nível de servidor. Contudo, por considerar o posicionamento em AZs diferentes, então a taxa $\mathbb{A}$ é a observada a nível da AZ.

Através da análise de arquiteturas de Data Centers (DCs) centralizadas, considera-se o mesmo servidor de processamento como o pior caso de um SPoF, e no melhor caso, uma arquitetura distribuída de DC com diferentes AZs, estendendo-se, em teoria, a Pro-

vedores de Serviços em Nuvem (CSPs) diferentes. Considera-se que um CSP pode ter sua organização distribuída em uma ou mais regiões geográficas, cada região composta por mais de três infraestruturas físicas independentes, *i.e.*, DCs isolados e compostos por múltiplas AZs com enlaces de rede de baixa latência entre si. Estas AZs são constituídas por componentes computacionais (rede, armazenamento distribuído, servidores de controle e processamento) e componentes de suporte (fontes de energia e de refrigeração, fonte de energia ininterrupta (UPS), unidade de distribuição de energia (PDU), etc.). Assim, sob a análise de RBD, as AZs são componentes em paralelo e independente de falhas de outras AZs, sustentando a característica de isolamento e minimizando o SPoF, essencial para promover a alta disponibilidade.

4.1. Definição formal da solução

Há duas perspectivas a serem consideradas: a do CSP e a do desenvolvedor. Sob a perspectiva do CSP, cada uma das p AZs são constituídas por m servidores que, em um determinado instante de tempo t, hospedam um total de n MVs, conforme ilustrado na Figura 1.

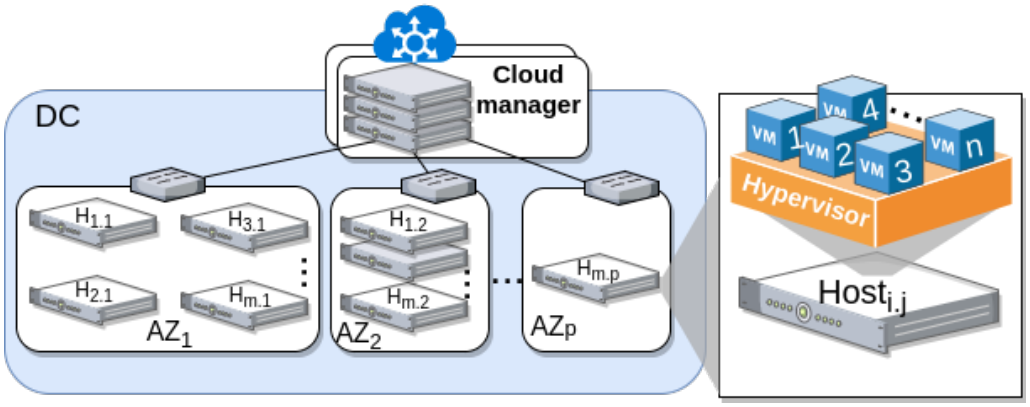

Figura 1. Perspectiva do CSP.

Assim, denota-se uma determinada MV em um DC por $\mathcal{V}_{ijk}$, em que $\mathcal{V}$ é o conjunto total de MVs que estão instanciadas em um instante t, com $0 \leq i \leq n$, $0 \leq j \leq m$ e $0 \leq k \leq p$. Este conjunto $\mathcal{V}_{nmp}$ é formado por subconjuntos de MVs críticas ($\mathcal{V}^c_{ijk}$), MVs réplicas ($\mathcal{V}^r_{ijk}$) e as demais MVs regulares. Cada um dos m servidores estão organizados em p AZs, e são denotados por $\mathcal{H}_{mp}$. Considerando servidores homogêneos, todos os elementos possuem o mesmo atributo, *i.e.*, mesma capacidade de CPU (em unidades) e RAM (em Gigabytes). Por outro lado, ao considerar servidores heterogêneos cada elemento tem os seus próprios atributos de recursos, como $\mathcal{H}_{mk} = \{h_1(cpu, ram), h_2(cpu, ram), ..., h_m(cpu, ram)\}$. Todos os símbolos e relações de elementos e conjuntos estão especificados na Tabela 2.

Da perspectiva de um desenvolvedor, especificada na Figura 2, podem haver p projetos, que são conjuntos de instâncias que, a nível de Infraestrutura como Serviço (IaaS), incluem MVs, redes virtuais, volumes de armazenamento, etc. Para a presente proposta considera-se como instâncias apenas as MVs, partindo-se do princípio que redes e armazenamento são gerenciados e provisionados pelo orquestrador da plataforma de computação em nuvem. Em cada projeto, há um conjunto com n MVs ($\mathcal{V}_n$), em que x são

consideradas como críticas ($\mathcal{V}_x^c$) e as demais são regulares. O nível de criticidade de cada MV é estabelecida por uma taxa $\mathbb{HA}$, especificada pelo desenvolvedor individualmente a cada MV. As demais MVs não especificadas por $\mathbb{HA}$, ou que requerem uma taxa menor que a disponibilidade regular $\mathbb{A}$, mantém-se sob a taxa de disponibilidade regularmente especificada em Acordo de Nível de Serviço (SLA).

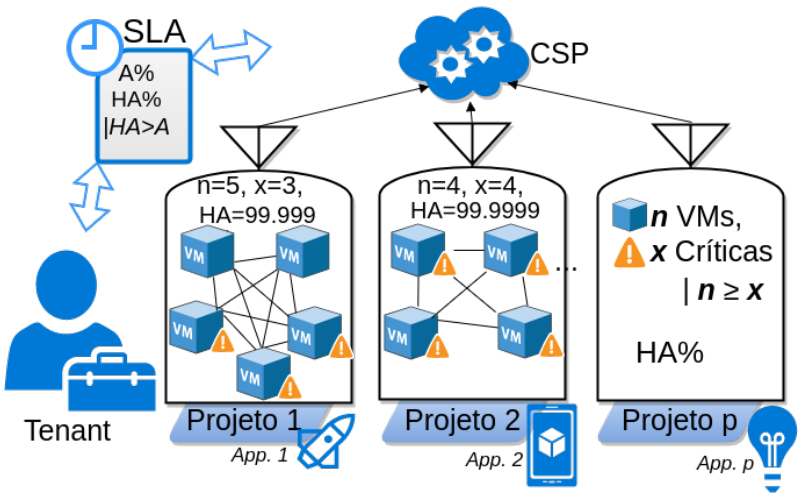

Figura 2. Perspectiva do desenvolvedor.

Notação	Legenda
	Conjuntos de recursos
$\mathcal{V}_{nmp}$	Conjunto de todas as n MVs alocadas em m servidores em p AZs
$\mathcal{V}_n$	Conjunto de n MVs em requisição para instanciação
$\mathcal{V}_x^c \mid \mathcal{V}_x^c \supseteq \mathcal{V}_n,\ x \leq n$	Conjunto de x MVs críticas
$\mathcal{V}_u^{H_j}$	Conjunto de todas as u MVs alocadas em um servidor j
$\mathcal{C}_i = v_i^c \bigcup \mathcal{V}_i^r \mid i \in x$	Conjunto de uma MV crítica, com suas respectivas réplicas (Equação 3)
$\mathcal{H}_{mp}$	Conjunto de m servidores em p AZs
$\mathcal{Z}_p$	Conjunto de p AZs
	Recursos físicos e virtuais
$R_n^{dem} = \sum_{i \in n}(CPU_i, RAM_i)$	Total da demanda de recursos de uma requisição $\mathcal{V}_n$ de MV a serem alocadas
$\mathcal{O}_{ju}^{'} = \left(\frac{\sum_{i=0}^{u} CPU_i}{CPU_j}, \frac{\sum_{i=0}^{u} RAM_i}{RAM_j}\right)$	Taxa de ocupação dos recursos das u MVs em um servidor j.
$\mathcal{O}_j^{''} = (100\%, 100\%) - \mathcal{O}_{ju}^{'}$	Taxa de recursos ociosos/disponíveis de um servidor j.
$R_k^{disp} = \sum_{j \in \mathcal{H}_m} \mathcal{O}_j^{''} \mid k \in \mathcal{Z}_p$	Total de recursos disponíveis em um conjunto $\mathcal{H}_m$ de servidores em p AZs.
$\sum_j^p \sum_i^m E_{ij}^{\Delta t}$	Consumo de m servidores em p AZs por um período Δt.

Tabela 2. Tabela de símbolos e relações utilizadas.

4.1.1. Estratégia de consolidação de MVs

A definição formal da consolidação de MVs pode ser dada através da programação inteira linear para o Problema do Bin Packing (PBP). Por definição, considera-se no PBP que há um conjunto infinito de *bins* para alocar todos os n *itens*, de modo a minimizar o

número de *bins* utilizados. Para a presente proposta, considera-se que para cada AZ k há um conjunto máximo de m servidores $\mathcal{H}_{mk} = \{h_1, h_2, \ldots, h_m\}$ homogêneos (mesma dimensão de recursos físicos $S = (cpu, ram)$), e uma lista de n MVs a serem alocadas $\mathcal{V}_n = \{v_1, v_2, \ldots, v_n\}$, cada MV com recursos de memória RAM e CPU, denotados por $v_i(vcpu, vram)$. Assim, deve-se encontrar o menor número de servidores L necessários para instanciar todos os elementos de $\mathcal{V}_n$. Isto é, $\sum_{i \in H_j} v_i \leq S$ para todo $j = 1, .., L$, observando-se que $L \leq m$ e, portanto, $H_L \subseteq H_m$. A solução é ótima se ela possui o mínimo L, sendo que o valor da solução ótima é denotado por OPT. A formulação para a programação inteira linear para o PBP é dado pelo bloco de Equações 4(a ao h). Adiciona-se duas restrições relacionadas a justaposição de MVs críticas e réplicas no mesmo servidor.

$$\text{Minimizar } L = \sum_{i=1}^{n} y_i \tag{4a}$$

$$\text{Sujeito a: } L \geq 1, \tag{4b}$$

$$\sum_{j \in \mathcal{V}_n} v_j.x_{ij} \leq S.y_i, \forall i \in \{1, ..., m\}, \tag{4c}$$

$$\sum_{j \in \mathcal{V}_n} x_{ij} = 1, \forall i \in \{1, ..., m\}, \tag{4d}$$

$$\sum_{k \in \mathcal{Z}_p} \sum_{j \in \mathcal{C}} x_{ijk} \leq 1, \forall i \in \{1, ..., m\}, \tag{4e}$$

$$\text{Decisão: } y_i \in \{0, 1\}, \forall i \in \{1, ..., m\}, \tag{4f}$$

$$x_{ij} \in \{0, 1\}, \forall i \in \{1, ..., m\}, \forall j \in \{1, ..., n\}, \tag{4g}$$

$$x_{ijk} \in \{0, 1\}, \forall i \in \{1, ..., m\}, \forall j \in \{1, ..., n\}, \forall k \in \{1, ..., p\} \tag{4h}$$

A função objetivo (4a) consiste em minimizar L, na primeira restrição (4b) especifica-se que haverá uma solução inteira positiva, na segunda (4c) restringe-se a alocação ao tamanho dos servidores e a terceira (4d) visa garantir que a mesma MV não seja atribuída a mais de um servidor. Com base em Speitkamp e Bichler 2010, podem ser adicionadas novas restrições visando atender as necessidades do modelo desejado. Deste modo, adiciona-se a restrição de justaposição (4e) para considerar arquiteturas Multi-AZs, em que $\mathcal{Z}_p$ é o conjunto de AZs possíveis de instanciar uma MV pertencente à $\mathcal{C}$. Por fim, são especificadas as variáveis de decisão em (4f): $y_i = 1$ se o servidor i é usado e em (4g e 4h) $x_{ij(k)} = 1$ se a MV j é instanciada no servidor i (e na AZ k).

Na prática, o processo de consolidação necessita ter uma tomada rápida de decisão, respondendo com a melhor configuração de alocação das n MV sempre que requisitada. Deste modo, a heurística *First Fit Decreasing* (FFD) adotada para o presente trabalho é selecionada pela relação entre critério de otimalidade e tempo de resposta. São propostas duas variações FFD para ambientes dinâmicos: (i) *FFD Dynamic Decreasing* (FF3D) que ordena em decrescente os recursos disponíveis de todos os servidores para alocar as MVs; e (ii) *FFD Dynamic Increasing* (FF2DI), que ordena os recursos disponíveis dos servidores em ordem crescente antes de alocar as MVs. O processo de consolidação de CHAVE é aplicado considerando uma arquitetura Multi-AZ, e executado individualmente para cada AZ através de uma perspectiva local. Esta aplicação individual torna-se importante porque os parâmetros de entrada são pertinentes a cada AZ. Estes parâmetros de entrada são:

- uma lista $\mathcal{V}_n$ com as n MVs, que ora são as requisições a serem alocadas na etapa de posicionamento, ora são todas as MVs já alocadas para a etapa de migração;
- uma lista $\mathcal{H}_m$ com m servidores já em execução, que ora são os servidores já em execução na etapa de posicionamento, ora é uma lista vazia de servidores na etapa de migração.

Considerar os recursos disponíveis em cada servidor, relacionando às suas MVs que já estão em execução é uma abordagem característica de ambientes dinâmicos. Entre os recursos disponíveis, ressalta-se que o processador pode ser avaliado pelo número de CPUs/núcleos ou capacidade de *hypertreading*, doravante considerados apenas como unidades de CPU. As unidades a memória RAM são avaliadas por unidade de gigabytes (GB), e são consideradas como um recurso restrito pelo escalonador da nuvem, já que não é comum às plataformas de nuvem aplicarem superutilização (*overbooking*) de memória.

CHAVE realiza o processo de consolidação de MVs em ambientes dinâmicos, que ocorre em duas etapas: a migração de MVs e o posicionamento inicial das requisições, sendo que em ambas as etapas é utilizado o mesmo algoritmo de consolidação (FF2DI e FF3D), mas alterando seus parâmetros de entrada. A migração de MVs é aplicada para consolidar a configuração esparsa de ambientes dinâmicos, decorrente da desalocação de MVs e liberação de recursos, sendo um processo que naturalmente ocorre durante o tempo. Os parâmetros de entrada da etapa de migração são: $\mathcal{V}_n$ é o conjunto de todas as MVs que já estão em execução na AZ, e ($\mathcal{H}_m$) todos os m servidores que já estão ativados. Como resultados no melhor caso, espera-se obter servidores ociosos, que podem ser desativados ou receber as novas requisições da etapa de posicionamento, e no pior caso, a consolidação resultará em uma minoria de servidores subutilizados e a maioria de servidores com 100% dos recursos utilizados ($\mathcal{O}_u^j = (100\%, 100\%)$). A etapa de posicionamento inicial de alocação recebe como entrada uma lista $\mathcal{V}_n$ das n requisições MVs que serão instanciadas e uma lista $\mathcal{H}_m$ com os m servidores que já foram consolidados na primeira etapa. Assim, nesta etapa espera-se como melhor caso, alocar todas as n requisições nos servidores subutilizados (resultado do pior caso na primeira etapa), e no pior caso ativar novos servidores para atender à demanda. Ativar novos servidores é naturalmente esperado quando o total de recursos da requisição VV_n (R_n^{dem}) é maior do que o total de recursos disponíveis nos servidores (R_k^{disp}) atualmente em execução. Porém, observa-se que a ordem destas etapas podem alterar as configurações finais, pois uma saída é a entrada de outro. Assim, no estudo de caso, especificado na Seção 5, São verificadas quais as possíveis consequências desta ordem. O pseudocódigo da heurística FF2DI/FF3D está descrito no Algoritmo 1.

```
ALGORITMO 1: FFDx() FF2DI / FF3D
Entradas: [<Hosts>:H, <MVs>:V, <Abordagem>:Ordem='FF2DI' ou 'FF3D']
Saída: Conjunto de hosts consolidados
01: ordena_decrescente(V)
02: Se Ordem é 'FF2DI', então
03:   ordena_crescente(H)
04: Senão Se Ordem é 'FF3D', então:
05:   ordena_decrescente(H)
06: Fim Se
07: Para cada elemento i de V até n faça
08:   Para cada elemento j de H até m, faça
09:     Se MV i.CPU cabe no host j, então
```

```
10:       Se MV i.RAM cabe no host j, então
11:         Aloque/Migre a MV i no/para host j
12:         Atualiza RAM e CPU disp. no host j
13:         Quebra o loop e analisa próxima MV
14:       Fim Se
15:     Fim Se
16:   Fim Para
17:   Se MV i não couber em nenhum host ativo, então
18:     Ative um novo host (j+1) e aloque  i
19:   Fim Se
20: Fim Para
21: Para cada elemento j de H
22:   Se j estiver ocioso, então DesativeHost(j)
23: Retorne <Hosts> H
```

Na primeira linha do Algoritmo 1, é organizada a lista de MVs de forma decrescente, *i.e.*, do maior para o menor tamanho de CPU e memória RAM, e na segunda linha é verificado qual a ordenação dos servidores de entrada, se é crescente (FF2DI) ou decrescente (FF3D). O processo de migrar MVs existentes ou alocar novas requisições de MVs ocorre na linha 11, permitindo que o mesmo algoritmo seja aplicado na primeira e na segunda etapa da consolidação. Ativar um novo servidor (linha 18) significa alterar um servidor do estado de desativado (hibernado ou desligado) para o estado ativo, sempre executado na etapa de posicionamento para atender à nova demanda de requisições. Normalmente o resultado da etapa de migração ocorre ao final do *loop* de execução (linha 22), em que se houver algum servidor ocioso, então este deve ser desativado, o que resulta na redução no consumo de energia elétrica. A saída do Algoritmo 1 será a nova configuração dos servidores fornecidos como parâmetro de entrada, que após executado na primeira e na segunda etapa, estarão devidamente consolidados. Este valor pode ser usado para obter uma taxa de consolidação TC de uma determinada AZ_i, através da Equação 5.

$$TC(AZ_i) = \frac{|\mathcal{H}_m^{final}| - |\mathcal{H}_m^{inicial}|}{|\mathcal{H}_m^{inicial}|} * 100\% \tag{5}$$

Utilizando-se da relação entre a saída do Algoritmo 1 e o conjunto de todos os servidores da AZ em questão, a Equação 5 define a razão entre os servidores ativos e o total da AZ como a taxa de consolidação atual $TC(AZ_i)$. Esta equação pode ser utilizada como um coeficiente para o balanceamento de carga entre todas as AZs, útil quando o desenvolvedor deixa em aberto a especificação da AZ que será instanciada suas MVs.

Com o objetivo de minimizar o número de migrações, o presente trabalho também propõe uma estratégia para evitar que hajam migrações desnecessárias na etapa de migração. Deste modo, é possível ter uma taxa de consolidação mas sem o custo de realizar diversas migrações para alcançar um objetivo similar. Esta estratégia é orientada a migração mínima (MMA – do inglês *minimum migration-aware*) e pode ou não ser aplicada ao Algoritmo 1.

4.1.2. Mecanismo de HA

Normalmente um CSP conta com uma arquitetura Multi-AZ, permitindo ao desenvolvedor especificar em qual AZ será instanciada cada uma de suas MVs. Por este motivo

a consolidação de MVs, nas etapas de posicionamento e de migração, deve manter esta consistência e serem executados localmente para cada AZ. Para as x MVs críticas que serão replicadas, deve-se para estabelecer o número de réplicas r para cumprir a taxa $\mathbb{HA}$ especificada pelo desenvolvedor. Para estabelecer r, CHAVE utiliza a Equação 3, que tem como parâmetros $\mathbb{HA}$ e $\mathbb{A}$, porém para obter $\mathbb{A}$, deve-se considerar onde serão distribuídas estas réplicas. Considerando a distribuição em uma arquitetura Multi-AZ, a disponibilidade $\mathbb{A}$ utilizada na Equação 3 será a mesma taxa especificada em SLA, pois compreende a disponibilidade de todos os componentes computacionais e de suporte contidos em cada AZ. Por outro lado, ao considerar a distribuição das réplicas em uma mesma zona, *e.g.*, em uma sala de servidores com diversos SPoF em comum, então a disponibilidade $\mathbb{A}$ deve ser calculada utilizando a Equação 1, obtida por análise histórica (tempo passado) ou probabilística (tempo médio entre falhas (MTBF) e tempo médio para reparo (MTTR)).

O presente trabalho parte do princípio de implementação da proposta em arquiteturas Multi-AZ, pois além de constituir um padrão entre os CSPs [Unuvar et al. 2014], é uma prática que possibilita simplificar mecanismos de HA. Deste modo, cada réplica será instanciada em uma AZ diferente da MV crítica e diferentes entre si, respeitando a restrição de justaposição. Observa-se que aplicar HA em uma arquitetura Multi-AZs limita o número de réplicas ao número de AZs. Porém, tomando-se como base CSPs públicas como *Amazon Web Services* (AWS), *Google Cloud Platform* (GCP) e Microsoft Azure, elas garantem sua resiliência como um mínimo de três zonas isoladas fisicamente em todas as regiões habilitadas. Assim, considerando como base uma taxa de disponibilidade $\mathbb{A}$ de 99,95% com duas réplicas, se obtém uma taxa de $\mathbb{HA}$ na ordem de seis noves. Para fornecer taxas ainda maiores, basta o CSP adicionar mais AZs para cada região.

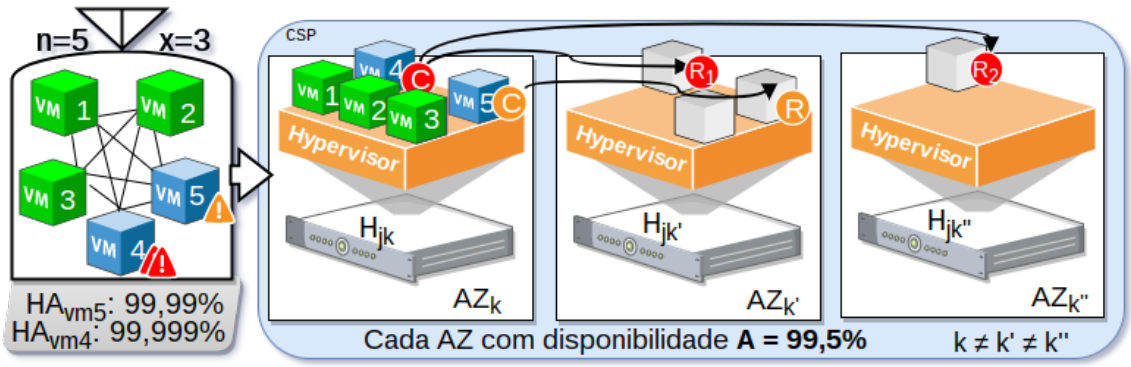

Figura 3. Visão geral da replicação em arquiteturas Multi-AZ.

Pode-se tomar como exemplo a Figura 3, uma MV crítica (v_4^c) com $\mathbb{HA}$=99,999%, em um CSP que estabelece $\mathbb{A}$=99,5% para todas as suas AZs. Assim, como resultado da Equação 3, exige-se mais duas réplicas, pois $r = \frac{log_{10}(1-0,99999)}{log_{10}(1-0,995)} - 1 = 1,17$ e $ceil(r) = 2$. A restrição de justaposição consiste em considerar um conjunto $\mathcal{C}_i$ com a MV crítica v_i^c e suas duas réplicas (v_1^r e V_2^r), de modo que quando v_i^c for instanciada na AZ_k, então suas respectivas réplicas deverão ser instanciadas em $AZ_{k'}$ e $AZ_{k''}$, de modo que $k \neq k' \neq k''$. Ao realizar o processo de consolidação para cada uma destas k AZs, topologia de distribuição de réplicas restringe alocar quaisquer elementos de $\mathcal{C}_i$ no mesmo SPoF.

5. Estudo de caso

O presente trabalho aplica as diferentes abordagens previamente apresentadas em um estudo de caso, com o objetivo compreender como ocorre a distribuição das máquinas virtuais (MVs) nos servidores de uma determinada zona de disponibilidade (AZ) e quais as relações podem ser adotadas entre si para atingir os melhores resultados. Os experimentos são inicialmente realizados através de simulação, para abstrair a alta complexidade de um ambiente de nuvem. O simulador utilizado é uma versão modificada do Eavira [Ruck et al. 2017], pois é utilizado em pesquisas relacionadas ao escalonamento e migração de Infraestruturas Virtuais (IVs) em ambientes de computação em nuvem e foi adaptado para receber apenas MVs.

As entradas de dados são primeiramente geradas pelo Método de Monte Carlo simplificado para obter dados aleatórios, como as requisições $\mathcal{V}$ e os servidores em uso $\mathcal{H}$. Em um segundo momento, são utilizados *traces* reais de carga de trabalho de um conjunto de seis diferentes AZs de uma nuvem Infraestrutura como Serviço (IaaS) privada, da plataforma Eucalyptus [Wolski e Brevik 2014]. O algoritmo do simulador é implementado em linguagem Python 3, os testes foram executados em um computador com processador AMD Phenom II X4, 4 GB RAM e sistema operacional Ubuntu 16.04.

Nestas entradas de dados, considera-se que o desenvolvedor especifica em quais AZs devem ser instanciadas suas MVs. Portanto, para manter a consistência deve-se instanciar apenas na AZ determinada pelo desenvolvedor, mantendo as migrações de instâncias dentro da mesma AZs. Mesmo nos casos em que o desenvolvedor seleciona uma AZ aleatoriamente, oportunizando o balanceamento de carga inter-AZs, a alocação das requisições $\mathcal{V}$ ocorre por uma perspectiva local à cada AZ, pois seu algoritmo de alocação baseia-se na análise de dois recursos restritos no conjunto de servidores de cada AZ: o número de CPUs (*núcleos*) e quantidade de memória RAM. Desta forma, para o presente estudo de caso são desconsiderados os recursos de rede e de armazenamento, pois são recursos gerenciados pelo orquestrador da nuvem, externamente aos servidores.

Adicionalmente, considera-se que cada AZ é constituída por servidores homogêneos, com mesmo número de CPUs e RAM em potência de base dois ($2^x | 0 \leq x \leq MAX = \{1, 2, 4, 8, 16, ..., 2^{MAX}\}$), mantendo-se uma proporção fixa entre ***CPU:RAM*** dos servidores para os recursos de ***vCPU:vRAM*** dos sabores das MVs. Um sabor define a dimensão dos recursos computacionais (CPU virtual e memória RAM virtual) que devem ser alocados para cada MV. Normalmente, para maximizar a capacidade de alocação dos recursos físicos, os sabores são definidos na mesma proporção que os recursos dos servidores. Se a proporção de CPU:RAM for de 1:2 de um servidor com 32 núcleos então este deverá ter 64GB de RAM, e os sabores das MVs deverão manter as proporções de 1:2, 2:4, 4:8, etc para vCPU:vRAM. Para aproximar de um ambiente real, são utilizados os quatro sabores da família m3 da *Amazon Web Services* (AWS) [Persico et al. 2015], que oferecem uma proporção de 1:3,75, e variam de um a oito vCPUs. Esta abordagem além de maximizar a alocação dos recursos físicos, facilita a análise para a consolidação de MVs.

Para realizar a consolidação de MVs, considera-se uma arquitetura Multi-AZ para garantir que não haja justaposição entre uma MV crítica e suas respectivas réplicas. Assim, o mecanismo de alta disponibilidade (HA) torna-se agnóstico à estratégia de consolidação e permite que diversas abordagens possam ser implementadas sem afetar

a HA requisitada.

5.1. Plano de testes

Entre estas abordagens, estão as relacionadas a ordem entre o posicionamento e a migração, na seleção entre o FF2DI ou o FF3D e usando ou não uma estratégia de migração mínima. Deste modo, a Tabela 5.1 relaciona as todas as abordagens possíveis de serem realizadas.

Estratégia	**Posiciona-migra**	**migra-posiciona**	**FF2DI**	**FF3D**	**MMA**
E_0	*Sem consolidação*				
E_1	✓		✓		
E_2	✓		✓		✓
E_3	✓			✓	
E_4	✓			✓	✓
E_5		✓	✓		
E_6		✓	✓		✓
E_7		✓		✓	
E_8		✓		✓	✓

Tabela 3. Relação das estratégias realizadas.

Os nove testes realizados suportam todas as abordagens discutidas na Seção anterior. Uma verificação inicial (S_0) é feita sem a consolidação, e é utilizada uma abordagem de instanciação aleatória, com vetor de requisições $\mathcal{V}$ na ordem em que foi gerado/lido. Para as demais simulações, é selecionada apenas uma estratégia entre a segunda coluna (posiciona-migra) e a terceira coluna (migra-posiciona), uma estratégia entre a quarta (FF2DI) e a quinta (FF3D) coluna, e uma seleção binária para a sexta coluna (MMA). Deste modo, a combinação entre as estratégias habilita a análise nos fatores de desempenho, eficiência e tempo de resposta de cada algoritmo. Objetiva-se com isso, em cada execução usar as mesmas entradas: vetor de requisições $\mathcal{V}$ e conjunto de servidores em execução $\mathcal{H}$ Para cada estratégia, serão executadas com dois conjuntos de dados diferentes: gerações aleatórias e os *traces* do Eucalyptus. Nas gerações aleatórias, serão realizadas 100.000 execuções para cada uma das estratégias.

Configurações adicionais: Overbooking

No *trace* do Eucalyptus há em diversas requisições uma superutilização dos servidores quando há mais demanda de CPUs virtuais do que a quantidade de núcleos previamente estabelecido. Nesse sentido, essas ações são consideradas como *overbooking* ou *overcommitting*, que foram detectadas a um nível de até 2468.75 %, *i.e.*, o total de instâncias em um único servidor ultrapassa em 24 vezes a quantidade de núcleos lógicos especificados por padrão. Deste modo, é adicionada ao simulador a funcionalidade de instanciar mais máquinas que a limitação de núcleos em um servidor. Para os testes realizados com as duas entradas de dados, são consideradas as duas possibilidades: habilitar ou não o *overbooking*. Quando habilitado, o *overbooking* é configurado com uma taxa limite de 2400%, e quando desabilitado, as requisições de instanciação que ultrapassam essa taxa são consideradas como violações de SLA, por negar a alocação da requisição.

Para controlar a entrada de dados dos *traces* é utilizado o conceito de janelas, que são filas de requisições de tamanho n e submetidas a cada ciclo de tempo. Assim, a configuração das janelas baseia-se em duas informações: o tamanho da janela e o tempo entre janelas, consideradas como duas variáveis de controle. Enquanto o tamanho da janela consiste em controlar a quantidade de requisições de MVs em cada janela, o tempo entre janelas consiste em controlar o fluxo da entrada de dados, permitindo simular um temporizador compatível com o *timestamp* configurados nos *traces*. Esse procedimento é indispensável por se tratar de algoritmo *offline*, que demandam ter consciência da fila para tomada de decisão. Quando estas a informações do tamanho da janela possui tamanho unitário, a instanciação é realizada de modo *online*, descartando qualquer ação do algoritmo *offline*. Deste modo, por não haver conhecimento da informação da janela nos *traces* do Eucalyptus, são testadas diversas configurações do tamanho da janela de 20 a 100 unidades (em passo 20), e o tempo da janela de 20 a 120 unidades de tempo (passo 20).

Para realizar a migração, além de usar uma janela de tempo similar a da fila de requisições, também e adicionado um gatilho acionado quando ha uma determinada taxa de fragmentação dos recursos físicos do DC. Dessa forma, garante-se que apenas a janela de tempo seja uma abordagem insuficiente para evitar que essa taxa de consolidação varie conforme a demanda de requisições. essa taxa é determinada pela razão entre a soma de todos os recursos que estão disponíveis nos servidores ativos pelo total de recursos físicos de apenas um servidor, quando homogêneos, ou do menor de seus servidores, quando heterogêneos. Desse modo, é possível verificar se, ao realizar o processo de migração, será permitido desativar ao menos um servidor.

6. Análise dos resultados

Os testes estão em fase de finalização e serão concluídos em sete dias!

7. Considerações finais

Alinhar uma estratégia de consolidação ao mecanismo de alta disponibilidade (HA) possibilita reduzir o impacto do consumo de energia na infraestrutura física da nuvem. Desse modo, é possível prover serviços que requerem alta disponibilidade com um mínimo adicional de consumo de energia. O mecanismo de HA baseado em replicação passiva permite que serviços com características *stateful* mantenham-se continuamente em execução, com réplicas instanciadas em múltiplas zona de disponibilidade (AZ) para reduzir o ponto único de falhas (SPoF) Os testes realizados foram desenvolvidos para habilitar uma análise de todas as nove possibilidades, e verificar em quais configurações é possível obter melhores resultados.

Referências

[Abdelsamea et al. 2017] Abdelsamea, A., El-Moursy, A. A., Hemayed, E. E., e Eldeeb, H. (2017). Virtual machine consolidation enhancement using hybrid regression algorithms. *Egyptian Informatics Journal*.

[Ahmad 2015] Ahmad, R. W. *et. al*. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *JNCA*, 52:11–25.

[Alahmadi et al. 2014] Alahmadi, A., Alnowiser, A., Zhu, M. M., Che, D., e Ghodous, P. (2014). Enhanced First-Fit Decreasing Algorithm for Energy-Aware Job Scheduling in Cloud. In *2014 International Conference on Computational Science and Computational Intelligence*, volume 2, pages 69–74.

[Alkawsi et al. 2015] Alkawsi, G. A., Mahmood, A. K., e Baashar, Y. M. (2015). Factors influencing the adoption of cloud computing in SME: A systematic review. In *Mathematical Sciences and Computing Research (iSMSC), International Symposium on*, pages 220–225. IEEE.

[Amoon 2016] Amoon, M. (2016). Adaptive Framework for Reliable Cloud Computing Environment. *IEEE Access*, 4:9469–9478.

[Balalaie et al. 2016] Balalaie, A., Heydarnoori, A., e Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, 33(3):42–52.

[Bauer e Adams 2012] Bauer, E. e Adams, R. (2012). *Reliability and Availability of Cloud Computing*. Wiley-IEEE Press, Piscataway, NJ : Hoboken, NJ, 1 edition edition.

[Beloglazov 2013] Beloglazov, A. (2013). Energy-efficient management of virtual machines in data centers for cloud computing. Phd thesis, The University Of Melbourne.

[Beloglazov e Buyya 2012] Beloglazov, A. e Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency Computat.: Pract. Exper.*, 24(13):1397–1420.

[Beyer et al. 2016] Beyer, B., Petoff, J., Jones, C., e Murphy, N. R. (2016). *Site Reliability Engineering*. O'Reilly, Sebastopol, CA, 1st edition.

[Bias 2016] Bias, R. (2016). The History of Pets vs Cattle and How to Use the Analogy Properly. *Cloudscaling website, (04/2016)*.

[Bin et al. 2011] Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E. K., Moatti, Y., e Lorenz, D. H. (2011). Guaranteeing High Availability Goals for Virtual Machine Placement. In *2011 31st International Conference on Distributed Computing Systems*, pages 700–709.

[Burns 2017] Burns, S. (2017). High availability in cloud computing prevents a SPOF. *TechTarget: SearchITOperations (07/2017)*.

[Comerford 2015] Comerford, T. (2015). How data center operators can avoid energy price hikes this winter.

[Corradi et al. 2012] Corradi, A., Fanelli, M., e Foschini, L. (2012). Vm consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, 32:118–127.

[Coulouris et al. 2011] Coulouris, G., Dollimore, J., Kindberg, T., e Blair, G. (2011). *Distributed Systems: Concepts and Design*. International computer science series. Pearson, Boston, 5 edition edition.

[Critchley 2014] Critchley, T. (2014). *High Availability IT Services*. CRC Press.

[Cully et al. 2008] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., e Warfield, A. (2008). Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco.

[David 2014] David, B. (2014). *Benchmarking, Consistency, Distributed Database Management Systems, Distributed Systems, Eventual Consistency*. KIT Scientific Publishing.

[Dong et al. 2013] Dong, Y., Ye, W., Jiang, Y., Pratt, I., Ma, S., Li, J., e Guan, H. (2013). COLO: COarse-grained LOck-stepping virtual machines for non-stop service. *ACM Press*, pages 1–16.

[Dósa et al. 2013] Dósa, G., Li, R., Han, X., e Tuza, Z. (2013). Tight absolute bound for First Fit Decreasing bin-packing: FFD (L) $\leq$ 11/9 OPT (L) + 6/9. *Theoretical Computer Science*, 510:13–61.

[Endo et al. 2016] Endo, P. T., Rodrigues, M., Gonçalves, G. E., Kelner, J., Sadok, D. H., e Curescu, C. (2016). High availability in clouds: systematic review and research challenges. *Journal of Cloud Computing*, 5:16.

[Ferdaus et al. 2014] Ferdaus, M. H., Murshed, M., Calheiros, R. N., e Buyya, R. (2014). Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic. In *Euro-Par 2014 Parallel Processing*, Lecture Notes in Computer Science, pages 306–317. Springer, Cham.

[Fischer e Mitasch 2006] Fischer, W. e Mitasch, C. (2006). High availability clustering of virtual machines–possibilities and pitfalls. *Paper for the talk at the 12th Linuxtag, May 3rd-6th, Wiesbaden/Germany Version*, 1.

[Ford et al. 2010] Ford, D., Labelle, F., Popovici, F., Stokely, M., Truong, V.-A., Barroso, L., Grimes, C., e Quinlan, S. (2010). Availability in Globally Distributed Storage Systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*.

[Hameed et al. 2016] Hameed, A., Khoshkbarforoushha, A., Ranjan, R., Jayaraman, P. P., Kolodziej, J., Balaji, P., Zeadally, S., Malluhi, Q. M., Tziritas, N., Vishnu, A., Khan, S. U., e Zomaya, A. (2016). A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774.

[Helal et al. 2006] Helal, A. A., Heddaya, A. A., e Bhargava, B. B. (2006). *Replication Techniques in Distributed Systems*. Springer Science & Business Media. Google-Books-ID: uybrBwAAQBAJ.

[Islam e Manivannan 2017] Islam, T. e Manivannan, D. (2017). Predicting Application Failure in Cloud: A Machine Learning Approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 24–31.

[Kanso et al. 2017] Kanso, A., Deixionne, N., Gherbi, A., e Moghaddam, F. F. (2017). Enhancing OpenStack Fault Tolerance for Provisioning Computing Environments. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pages 77–83.

[Kominos et al. 2017] Kominos, C., Seyvet, N., e Vandikas, K. (2017). Bare-metal, virtual machines and containers in OpenStack. In *20th Conference on Innovations in Clouds, Internet and Networks (ICIN), 2017*, pages 36–43.

[Koslovski et al. 2010] Koslovski, G., Yeow, W. L., Westphal, C., Huu, T. T., Montagnat, J., e Vicat-Blanc, P. (2010). Reliability support in virtual infrastructures. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 49–58.

[Li et al. 2016] Li, X., Qi, Y., Chen, P., e Zhang, X. (2016). Optimizing Backup Resources in the Cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 790–797.

[Machida et al. 2010] Machida, F., Kawato, M., e Maeno, Y. (2010). Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, pages 32–39.

[Maciel 2016] Maciel, P. R. M. (2016). Modeling Availability Impact in Cloud Computing. In *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering, pages 287–320. Springer, Cham. DOI: 10.1007/978-3-319-30599-8 11.

[Madni et al. 2016] Madni, S. H. H., Latiff, M. S. A., Coulibaly, Y., e Abdulhamid, S. M. (2016). Recent advancements in resource allocation techniques for cloud computing environment: a systematic review. *Cluster Computing*.

[Marston et al. 2011] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., e Ghalsasi, A. (2011). Cloud computing — The business perspective. *Decision Support Systems*, 51(1):176–189.

[Martello e Toth 1990] Martello, S. e Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70.

[Masakari 2017] Masakari (2017). Masakari - OpenStack.

[Matos et al. 2017] Matos, R., Dantas, J., Araujo, J., Trivedi, K. S., e Maciel, P. (2017). Redundant Eucalyptus Private Clouds: Availability Modeling and Sensitivity Analysis. *Journal of Grid Computing*, 15(1):1–22.

[Medina e García 2014] Medina, V. e García, J. M. (2014). A Survey of Migration Mechanisms of Virtual Machines. *ACM Comput. Surv.*, 46(3):30:1–30:33.

[Mell e Grance 2011] Mell, P. M. e Grance, T. (2011). SP 800-145. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States.

[Milani e Navimipour 2016] Milani, A. S. e Navimipour, N. J. (2016). Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. *Journal of Network and Computer Applications*, 71:86–98.

[Mondal et al. 2016] Mondal, S., Muppala, J., e Machida, F. (2016). Virtual Machine Replication on Achieving Energy-Efficiency in a Cloud. *Electronics*, 5(3):37.

[Moreno-Vozmediano et al. 2017] Moreno-Vozmediano, R., Montero, R. S., Huedo, E., e Llorente, I. M. (2017). Orchestrating the Deployment of High Availability Services on

Multi-zone and Multi-cloud Scenarios. *Journal of Grid Computing*, pages 1–15. DOI: 10.1007/s10723-017-9417-z.

[Muchalski e Maziero 2014] Muchalski, F. e Maziero, C. (2014). Alocação de Máquinas Virtuais em Ambientes de Computação em Nuvem Considerando o Compartilhamento de Memória. In *XII Workshop de Computação em Clouds e Aplicações - WCGA 2014*, pages 81–92, Florianópolis/SC–Brazil. XII WCGA 2014.

[Nanduri et al. 2014] Nanduri, R., Kakadia, D., e Varma, V. (2014). Energy and SLA aware VM Scheduling. *arXiv:1411.6114 [cs]*. arXiv: 1411.6114.

[Nathuji e Schwan 2007] Nathuji, R. e Schwan, K. (2007). VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 265–278, New York–NY, USA. ACM.

[Nguyen e Lebre 2017] Nguyen, T. L. e Lebre, A. (2017). Virtual Machine Boot Time Model. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 430–437. IEEE.

[Patel e Vaghela 2016] Patel, S. D. e Vaghela, D. (2016). A survey paper on load balancing algorithms for resource management in virtualization. *International Journal For Innovative Research In Science And Technology*, 2(11):68–70.

[Persico et al. 2015] Persico, V., Marchetta, P., Botta, A., e Pescapè, A. (2015). Measuring Network Throughput in the Cloud. *Comput. Netw.*, 93(P3):408–422.

[Petrovic e Schiper 2012] Petrovic, D. e Schiper, A. (2012). Implementing Virtual Machine Replication: A Case Study Using Xen and KVM. In *Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, AINA '12, pages 73–80, Washington, DC, USA. IEEE Computer Society.

[Prathiba e Sowvarnica 2017] Prathiba, S. e Sowvarnica, S. (2017). Survey of failures and fault tolerance in cloud. In *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, pages 169–172.

[Ranjan et al. 2015] Ranjan, R., Benatallah, B., Dustdar, S., e Papazoglou, M. P. (2015). Cloud Resource Orchestration Programming: Overview, Issues, and Directions. *IEEE Internet Computing*, 19(5):46–56.

[Riasetiawan et al. 2015] Riasetiawan, M., Ashari, A., e Endrayanto, I. (2015). Distributed Replicated Block Device (DRDB) implementation on cluster storage data migration. In *2015 International Conference on Data and Software Engineering (ICoDSE)*, pages 93–97.

[Rieck 2010] Rieck, B. (2010). Basic analysis of bin-packing heuristics.

[Ruck et al. 2017] Ruck, D., Miers, C., Pillon, M. A., e Koslovski, G. (2017). EAVIRA: Energy-Aware Virtual Infrastructure Reallocation Algorithm. In *Proceedings SBESC 2017*, page 14, Cutiba/PR/Brazil.

[Santos et al. 2017] Santos, G. L., Endo, P. T., Gonçalves, G., Rosendo, D., Gomes, D., Kelner, J., Sadok, D., e Mahloo, M. (2017). Analyzing the IT subsystem failure impact on availability of cloud services. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 717–723.

[Simonin et al. 2013] Simonin, M., Feller, E., Orgerie, A. C., Jégou, Y., e Morin, C. (2013). An Autonomic and Scalable Management System for Private Clouds. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 198–199.

[Speitkamp e Bichler 2010] Speitkamp, B. e Bichler, M. (2010). A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers. *IEEE Transactions on Services Computing*, 3(4):266–278.

[Stapelberg 2009] Stapelberg, R. F. (2009). *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design*. Springer Science & Business Media.

[Tamura et al. 2008] Tamura, Y., Sato, K., Kihara, S., e Moriai, S. (2008). Kemari: Virtual machine synchronization for fault tolerance using DomT. *Xen Summit 2008, USENIX ATC '08*.

[Unuvar et al. 2014] Unuvar, M., Doganata, Y., Steinder, M., Tantawi, A., e Tosi, S. (2014). A Predictive Method for Identifying Optimum Cloud Availability Zones. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 72–79.

[Villamizar et al. 2015] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., e Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590.

[Wang e Li 2015] Wang, Y. e Li, X. (2015). Achieve high availability about point-single failures in OpenStack. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, volume 01, pages 45–48.

[Weibull 2007] Weibull (2007). Reliability basics: Availability and the different ways to calculate it. *Weibull: Reliability HotWire*, 79.

[Wolski e Brevik 2014] Wolski, R. e Brevik, J. (2014). Using Parametric Models to Represent Private Cloud Workloads. *IEEE Transactions on Services Computing*, 7(4):714–725.

[Wu e Buyya 2015] Wu, C. e Buyya, R. (2015). *Cloud Data Centers and Cost Modeling: A Complete Guide To Planning, Designing and Building a Cloud Data Center*. Morgan Kaufmann.

[Xu et al. 2016] Xu, M., Tian, W., e Buyya, R. (2016). A survey on load balancing algorithms for vm placement in cloud computing. *Distributed, Parallel, And Cluster Computing (cs.dc)*.

[Yang et al. 2011] Yang, C. T., Chou, W. L., Hsu, C. H., e Cuzzocrea, A. (2011). On Improvement of Cloud Virtual Machine Availability with Virtualization Fault Tolerance Mechanism. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 122–129.

[Zhang et al. 2017] Zhang, Q., Liu, S., Qin, S., e Shi, Y. (2017). A column generation-based algorithm for two-stage, two-dimensional bin packing problem with a variant variable sized constraint. In *2017 36th Chinese Control Conference (CCC)*, pages 2841–2845.

[Zhang et al. 2014] Zhang, Q., Metri, G., Raghavan, S., e Shi, W. (2014). Rescue: An energy-aware scheduler for cloud environments. *Sustainable Computing: Informatics And Systems*, 4(4):215–224.

[Zhang et al. 2015] Zhang, X., Gaddam, S., e Chronopoulos, A. T. (2015). Ceph Distributed File System Benchmarks on an Openstack Cloud. In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 113–120.

[Zhou et al. 2017] Zhou, A., Sun, Q., e Li, J. (2017). Enhancing reliability via checkpointing in cloud computing systems. *China Communications*, 14(7):1–10.