UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DAMARES DA SILVA CAVALCANTE

PROCESSO DE TOMADA DE DECISÃO PARA GERENCIAMENTO DE ALOCAÇÃO E MIGRAÇÃO DE MÁQUINAS VIRTUAIS BASEADO EM CRITÉRIOS DE CONFIABILIDADE

CAMPINA GRANDE - PB
2023

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Processo de tomada de decisão para gerenciamento de alocação e migração de máquinas virtuais baseado em critérios de confiabilidade

Damares da Silva Cavalcante

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Computação

Reinaldo Cézar de Morais Gomes
(Orientador)

Campina Grande, Paraíba, Brasil

C376p Cavalcante, Damares da Silva.
Processo de tomada de decisão para gerenciamento de alocação e migração de máquinas virtuais baseado em critérios de confiabilidade / Damares da Silva Cavalcante. - Campina Grande, 2023.
67 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2023.
"Orientação: Prof. Dr. Reinaldo Cézar de Morais Gomes."
Referências.

1. Computação em Nuvem. 2. Integridade dos Ambientes. 3. Confiabilidade em Ambientes de *cloud computing*. 4. *Cluster* físico. 5. *Cluster* virtual. I. Gomes, Reinaldo Cézar de Morais. II. Título.

CDU 004(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA ITAPUANA SOARES DIAS GONÇALVES CRB-15/093

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
POS-GRADUACAO CIENCIAS DA COMPUTACAO
Rua Aprigio Veloso, 882, - Bairro Universitario, Campina Grande/PB, CEP 58429-900

FOLHA DE ASSINATURA PARA TESES E DISSERTAÇÕES

DAMARES DA SILVA CAVALCANTE

PROCESSO DE TOMADA DE DECISÃO PARA GERENCIAMENTO DE ALOCAÇÃO E MIGRAÇÃO DE MÁQUINAS VIRTUAIS BASEADO EM CRITÉRIOS DE CONFIABILIDADE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação como pré-requisito para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: 23/02/2023

Prof. Dr. REINALDO CÉZAR DE MORAIS GOMES, Orientador, UFCG

Prof. Dr. ANDREY ELÍSIO MONTEIRO BRITO, Examinador Interno, UFCG

Prof. Dr. ANDERSON FABIANO BATISTA FERREIRA DA COSTA, Examinador Externo, IFPB

Documento assinado eletronicamente por **REINALDO CEZAR DE MORAIS GOMES**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 23/02/2023, às 16:39, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **Anderson Fabiano Batista Ferreira da Costa**, **Usuário Externo**, em 23/02/2023, às 20:31, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **ANDREY ELISIO MONTEIRO BRITO**, **PROFESSOR 3 GRAU**, em 24/02/2023, às 11:41, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Resumo

A confiabilidade é um fator crítico em sistemas computacionais. Em ambientes de *cloud computing* a confiabilidade pode ser um fator decisivo para o gerenciamento de máquinas virtuais dentro de um *cluster* físico ou virtual. A garantia da confiabilidade está diretamente ligada a fatores relacionados à integridade dos ambientes em que as máquinas virtuais estão hospedadas. Dado que um sistema não-íntegro pode comprometer toda a infraestrutura de uma rede de computadores e do *cluster* em que está localizada. A utilização do chip TPM na forma física ou virtual permite agregação do requisito de integridade à máquinas virtuais ou hospedeiras. Aliado à verificação de logs de eventos é possível utilizar um mecanismo de atestação de integridade que monitore esse requisito nos nós de um cluster. Pensando nisso, este trabalho propõe um processo de alocação e migração automática de máquinas virtuais dentro de um cluster baseado em critérios de confiabilidade. A utilização das informações geradas pelo mecanismo de atestação podem ser utilizadas de forma automatizada para a tomada de decisão para a alocação ou migração de máquinas virtuais dentro do *cluster*. Os processos também levam em consideração requisitos de software e hardware, realizando o gerenciamento tanto a nível de segurança quanto a nível de verificação dos recursos do *cluster*, tomando decisão do melhor hospedeiro para as máquinas virtuais a depender dos critérios e requisitos especificados. Para a validação do processo proposto foi realizada uma avaliação nos ambientes de virtualização vSphere e KVM. Os experimentos foram realizados considerando critérios de software, hardware e confiabilidade e as métricas de avaliação foram medidas a partir da perspectiva do uso do tempo e do consumo de recursos computacionais. Os resultados demonstram que o processo proposto facilita o gerenciamento de alocação e migração dentro do *cluster*, além de otimizar a tarefa do administrador de monitorar manualmente a integridade das máquinas criadas.

Palavras-chave: Integridade, Computação em nuvem, Confiabilidade, *Cluster*

Abstract

Reliability is a critical factor in computer systems. In cloud environments computing reliability can be a deciding factor for machine management virtual clusters within a physical or virtual cluster. The guarantee of reliability is directly linked to factors related to the integrity of the environments in which the virtual machines are hosted. Given that an unhealthy system can compromise the entire infrastructure of a network of computers and the cluster in which it is located. The use of the TPM chip in physical or virtual form allows the aggregation of the health requirement to the virtual machines or hostesses. Allied to the verification of event logs, it is possible to use a mechanism health attestation that monitors this requirement on the nodes of a cluster. thinking Therefore, this work proposes an automatic machine allocation and migration process virtual machines within a cluster based on reliability criteria. The use of information information generated by the attestation mechanism can be used in an automated way for decision making for the allocation or migration of virtual machines within the cluster. The processes also take into account software and hardware requirements, meeting management both at the security level and at the level of verification of the resources of the cluster, making the decision of the best host for the virtual machines depending on the specified criteria and requirements. For the validation of the proposed process, a evaluation in vSphere and KVM virtualization environments. The experiments were carried out considering the software, hardware and reliability criteria and the evaluation metrics were measured from the perspective of time use and resource consumption computational. The results demonstrate that the proposed process facilitates the management allocation and migration within the cluster, in addition to optimizing the task of the manually monitor the health of created machines.

Keywords: Integrity, Cloud Computing, Confiabilidade, Reliability, Cluster

Agradecimentos

Agradeço a mim, pela força e persistência para conclusão deste trabalho.

Agradeço aos meus pais, meus irmãos e minha vó pela paciência, carinho e apoio durante essa jornada.

Agradeço à minha parceira da vida Ana Thais, por todo amor, paciência e companheirismo.

Agradeço ao meu orientador professor Dr. Reinaldo Cézar, pela orientação, compreensão e otimismo, mesmo quando eu não cumpria com as expectativas.

Agradeço também aos demais professores que me acompanharam durante o mestrado, cujo os conhecimentos transmitidos foram essenciais para minha formação.

Agradeço ao LSD e as todos os colegas que pude compartilhar momentos desafiadores e de aprendizagem durantes os projetos.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES e ao povo brasileiro pelo custeio da bolsa durante metade da minha jornada e espero poder retribuir o apoio obtido ao longo dos anos.

Agradeço também à EMBRAPII pela oportunidade de poder participar de um projeto cujos conhecimentos também foram essenciais para o meu desenvolvimento acadêmico e profissional.

Agradeço a todos que de alguma forma fizeram parte da minha jornada.

Agradeço a Deus, que me deu força e resiliência para concluir este trabalho.

Conteúdo

Lista de Símbolos

TPM - *Trusted Platform Memory*

vTPM - *Virtual Trusted Platform Memory*

IMA - *Integrity Measurement Architecture*

WBCL - *Windows Boot Configuration Log*

TGC - *Trusted Computing Group*

VM - *Virtual Machine*

PCR - *Platform Configuration Register*

NASA - *National Aeronautics and Space Administration*

FAA - *Federal Aviation Administration*

VMM - *Virtual Machine Monitor*

IAAS - *Infrastructure As a Service*

KVM - *Kernel Virtual Machine*

QEMU - *Quick Emulator*

ESX - *Elastic Sky X*

ESXi - *Elastic Sky X Integrated*

API - *Application Programming Interface*

SDK - *Software Development Kit*

VBox - *VirtualBox*

MAC - *Macintosh*

RSA - *Rivest-Shamir-Adleman*

EVM - *Extended Verification Module*

RA - *Remote Attestation*

SPIRE - *SPIFFE Runtime Environment*

SPIFFE - *Secure Production Identity Framework For Everyone*

SVID - *SPIFFE Verifiable Identity Documents*

PKI - *Public Key Infrastructure*

CNCF - *Cloud Native Computing Foundation*

AK - *Attestation Key*

UUID - *Universally Unique Identifier*

EK - *Endorsement Key*

AIK - *Attestation Identity Key*

SPS - *Self-Protection Software*

CPU - *Central Processing Unit*

MacOS - *Macintosh Operation System*

SHA1 - *Secure Hash Algorithm 1*

SHA256 - *Secure Hash Algorithm 256*

SO - *Sistema Operacional*

CCC - *Confidential Computing Consortium*

BIOS - *Basic Input/Output System*

DRA - *Dynamic Random Acess Memory*

PCIe - *Peripheral Component Interconnect Express*

USB-C - *Universal Serial Bus Type-C*

SSH - *Secure Socket Shell*

IDE - *Integrated Development Environment*

JDK - *Java Development Kit*

SWTPM - *Software Trusted Platform Module*

OVMF - *Open Virtual Machine Firmware*

UEFI - *Unified Extensible Firmware Interface*

AMD - *Advanced Micro Devices*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização e Motivação

Nos últimos anos houve um crescimento acentuado no desenvolvimento de tecnologias geradoras de grandes volumes de dados como a internet das coisas e as mídias sociais [22], amparados pela tecnologia presente na computação em nuvem esses ambientes necessitam cada vez mais de maior poder de processamento e maior capacidade para armazenamento de dados complexos com um grau mais elevado de gerenciamento de integridade.

A integridade em sistemas computacionais torna-se um requisito crítico, ganhando mais importância e atenção com os diversos incidentes de segurança ocorridos, tornando-se necessário a utilização de uma combinação de técnicas para fornecer as melhores características a depender dos requisitos críticos de cada ambiente [5]. A garantia de integridade pode ser obtida de diversas formas, seja através do uso de criptografia, autenticação de usuário, *backup* de dados, redundância de sistemas, sistemas de detecção de intrusão e a ancoragem da verificação de integridade via hardware com o uso do TPM ou através da sua forma virtualizada, com a ancoragem em circuitos integrados, chamado de vTPM.

Um sistema não íntegro pode comprometer toda a infraestrutura de uma rede de computadores ou mesmo de um ambiente de *cloud*, permitindo acesso indevido à serviços críticos do ambiente [28]. A garantia de integridade a nível de hardware tem sido amplamente adotada através da utilização de um chip físico chamado de *Trusted Platform Memory (TPM)* que permite uma camada de segurança a nível de hardware. O chip TPM é um módulo de segurança de baixo custo que é tipicamente implementado como um circuito integrado resistente à vi-

olações [14], diferentemente de cartões inteligentes, o TPM está vinculado diretamente ao ambiente de hardware em que foi desenvolvido, criando uma camada de segurança adicional e permitindo a identificação em caso de violação.

No mercado, existem muitos ambientes informatizados que ainda não contam com o TPM físico no hardware de suas máquinas. Alternativamente a esse contexto, ao longo dos anos houve um aumento crescente no desenvolvimento de pesquisas e implementação de alternativas à essa ausência, desenvolvendo-se uma versão virtualizada do chip TPM, chamado na literatura de *Virtual Trusted Platform Memory (vTPM)*. O *vTPM* permite a virtualização do TPM físico e apesar de não proporcionar as mesmas capacidades de segurança do TPM, por não possuir um processo de ancoragem em hardware, o vTPM cria uma partição lógica dedicada à ancoragem de dados de integridade que permite uma camada a mais de segurança às aplicações onde a integridade de dados seja um fator crítico [30].

Em um ambiente de *cluster* é possível a incorporação do chip vTPM em máquinas virtuais e em máquinas hospedeiras. A utilização do chip TPM nos nós do *cluster* aliado a um mecanismo que permita o monitoramento contínuo da integridade nos nós possibilita a garantia de que dados críticos das máquinas virtuais não sejam acessados por pessoas não autorizadas. Assim, a integridade tratada neste trabalho refere-se a garantia de que as informações e dados de um sistema são precisos, consistentes e confiáveis. Ao falar que um sistema possui integridade estamos nos referindo a garantia de que os dados originais não foram corrompidos e os mecanismos de garantia de integridade não foram violados por pessoas ou sistemas externos não autorizados.

Com a possibilidade de adoção do vTPM em diversas máquinas dentro de um *cluster* ou um ambiente de *cloud computing*, e a constante necessidade de gerenciamento desse tipo de ambiente, torna-se necessário a utilização de algum mecanismo que possibilite a automatização de determinadas tarefas otimizando todo o processo de monitoramento de integridade nos nós do ambiente. Além disso, é possível consumir dados históricos do ambiente e tomar decisões para criação, migração e manutenção de novos hospedeiros e máquinas virtuais dentro do *cluster* a partir de critérios que podem ser pré-estabelecidos. Sistemas de apoio à tomada de decisão podem ser utilizados dentro desse contexto para desacoplar a dependência de uma pessoa física gerenciando os ambientes de *cluster* e executando tarefas monótonas como a criação de máquinas virtuais. Utilizando-se de sistemas de decisão, a criação dessas

máquinas virtuais podem acontecer de acordo com critérios estabelecidos previamente pelos usuários.

A confiabilidade em sistemas computacionais refere-se à capacidade de um sistema ou componente de software de realizar as suas funções de forma consistente e previsível ao logo do tempo, sem falhas ou interrupções. Nesse aspecto, um sistema confiável manteria um certo nível de desempenho, eficiência e disponibilidade sob condições normais de uso e em situações de falha. A confiabilidade entra como um requisito chave para manter a disponibilidade de serviços oferecidos por um sistema, em um ambiente de *cloud* poderíamos pensar na capacidade de migrar uma máquina virtual em que o *host* de origem foi comprometido.

Assim, através de uma ferramenta que permita a análise de dados gerados no *cluster* para tomar decisões quanto a criação, migração, remoção ou manutenção das máquinas, é possível otimizar processos cotidianos realizados por um administrador de ambientes de *cluster*, além de controlar os *hosts* que hospedam as máquinas virtuais, analisando-os também quanto aos aspectos de software, hardware e confiabilidade, proporcionando uma ferramenta de gerenciamento de alto nível com tomada de decisão a partir da coleta de dados dos *clusters*.

1.2 Delimitação do Problema da Pesquisa

A utilização de dados históricos gerados pelos sistemas e aplicações permitem uma maior otimização no gerenciamento de aplicações através da criação de rotinas de decisão que levem em consideração estes dados. Muitas vezes, dentro de um contexto organizacional têm-se sistemas que realizam as operações cruciais de um negócio, sejam registrando vendas, cadastrando novos clientes ou mesmo realizando o gerenciamento da infraestrutura de uma *cloud* privada. Todas essas operações estão constantemente gerando *logs* de dados que muitas vezes não são utilizados para agregar valor às aplicações daquele negócio ou analisar contextos futuros de vulnerabilidades.

Durante a Pandemia de COVID-19 houve uma intensificação da migração de atividades presenciais para ambientes digitais, com a utilização cada vez maior de plataformas de *streaming*, *meet* e utilização de suítes de escritório online [3]. O acontecimento fez emergir uma maior atenção à área tecnológica e a todas as soluções aplicáveis aos vários setores, sejam

eles educacionais, de saúde, negócio ou setores públicos.

A definição de *cloud computing* reflete um modelo para permitir acesso sob demanda a um conjunto compartilhado de recursos computacionais configuráveis como redes, servidores, armazenamento, banco de dados, software, inteligência, aplicativos ou mesmo serviços. Tais elementos podem ser provisionados para acesso e utilização por meio de uma interação pré-existente com o servidor [26]. A utilização de computação em nuvem permite a viabilidade de outras grandes aplicações como o *Big Data*, que por definição é um conjunto de dados maior e mais complexo que deriva principalmente de novas fontes de dados. Por possuir um alto volume de dados com um extensa variabilidade, os softwares gerenciadores de banco de dados tradicionais não consegue gerenciar os dados complexos do *big data* [11]. A vantagem da sua utilização e que o coloca em uma posição de destaque frente às novas tecnologias é a possibilidade de identificação e resolução de problemas de negócios e sistemas que antes eram mais difíceis ou até mesmo inviáveis.

Dada a importância do fator integridade em ambientes de computação em nuvem, manter não apenas a integridade dos dados mas de todos os nós dentro de uma infraestrutura de *cloud* permite um nível mais alto de segurança, sendo possível realizar um monitoramento contínuo baseado em critérios definidos pelos usuários alocadores de nós no ambiente ou pelo analista da infraestrutura. Para manter o processo de monitoramento executando com base em eventos ocorridos torna-se necessário um componente que realize coletas periódicas de dados. Uma vez que os dados gerados por um nó podem ser utilizados para coleta, processamento, análise e interpretação para automatizar tarefas dentro do *cluster* como criação, migração e remoção de máquinas virtuais, além de contribuir preventivamente com a segurança através da identificação de possíveis vulnerabilidades no ambiente.

Neste trabalho, propomos um processo de tomada de decisão para gerenciamento de alocação e migração de máquinas virtuais baseado em critérios de confiabilidade. O processo será capaz de automatizar tarefas cotidianas como criação de máquinas virtuais seguindo requisitos pré-estabelecidos, bem como será capaz de manter ou remover máquinas virtuais que não atendem mais aos critérios de confiabilidade estabelecidos. Quanto ao *host*, o sistema será capaz de neutralizar a alocação de novas máquinas virtuais e torná-lo indisponível através da migração automática de todas as suas máquinas virtuais para outro *host* que atenda aos requisitos solicitados inicialmente no processo de alocação de VMs.

A ferramenta será implementada com base em subcomponentes que delegam determinadas atividades, como a atividade de coleta de dados e a atividade de atestação. Os componentes coletores serão desenvolvidos com o objetivo de coletar dados de *Platform Configuration Register (PCR)*, chaves de referência do chip TPM ou do vTPM e os *logs* de eventos, além de realizar requisições ao *cluster* para coleta dos recursos computacionais dos nós do ambiente. O componente de atestação será responsável por receber os valores de PCRs e as chaves de referência do TPM, realizar a autenticação do nó e validar os valores recebidos dos PCRs e retornar um status de verificação de autenticidade do nó, após essa verificação inicial ocorre a validação da integridade do nó através da verificação do arquivo de *log*, retornando um status final de *host íntegro* ou *não íntegro*. A integridade neste caso implicaria se houve ou não violação nos valores recebidos dos PCRs para os valores de referência armazenados.

A partir do processo de atestação fornecido por um componente externo são gerados dados dos nós do *cluster* que serão usados por este processo proposto, juntamente com os dados coletados de recursos computacionais do nós, para definir os hospedeiros das máquinas virtuais que estão em processo de alocação ou migração. A partir dos dados coletados, o componente de decisão irá verificar os dados recebidos e os requisitos e iniciar o processo de decisão de escolha do melhor *host* e do melhor *storage* - caso a opção de *storage* compartilhado tenha sido escolhida no momento da alocação.

1.3 Objetivos da Pesquisa

1.3.1 Objetivos Gerais

O presente trabalho tem por objetivo propor um processo de alocação e migração de máquinas virtuais levando em consideração critérios de hardware, software e confiabilidade. A solução proposta visa auxiliar o administrador de sistemas na realização de tarefas dentro da infraestrutura de uma *cloud*, otimizando o processo de alocação e migração de máquinas virtuais para outros hospedeiros dentro dos ambientes virtualizados com base em critérios pré-estabelecidos.

1.3.2 Objetivos Específicos

Dentro desse contexto, os seguintes objetivos específicos foram definidos:

- **Implementar um processo de alocação de máquinas virtuais:** Implementar um processo de alocação de máquinas virtuais com base em requisitos de software, hardware e de confiabilidade.
- **Implementar um processo de migração automática de máquinas virtuais:** Implementar um processo de migração automática de máquinas virtuais baseado em critérios de confiabilidade.
- **Implementar um processo de tomada de decisão:** Implementar um processo de tomada de decisão para alocação e migração automática de máquinas virtuais considerando critérios de confiabilidade.
- **Implementar uma prova de conceito para avaliar os processos propostos**: Implementar uma prova de conceito para avaliar o processo de alocação e migração baseada em critérios de confiabilidade.
- **Avaliar a interoperabilidade da solução proposta**: Avaliar a interoperabilidade da solução proposta.
- **Avaliar o desempenho e a integridade da solução proposta**: Avaliar o desempenho e a integridade da solução proposta considerando os processos de alocação e migração de máquinas virtuais.

1.4 Relevância e Contribuições

A computação em nuvem tornou-se um novo paradigma para compartilhamento de recursos e serviços dentro da *internet*. Dentro desse ambiente, a segurança na manutenção e compartilhamento de dados é fundamental, dado que os clientes devem confiar na infraestrutura em que estão construindo suas aplicações [21]. Em um ambiente de *cloud* comum, os usuários não têm controle físico sobre os seus dados, uma vez que podem ficar armazenados em qualquer nó na infraestrutura da empresa de hospedagem. Assim, fornecer um mecanismo

que permita ao usuário estabelecer parâmetros de confiabilidade no momento de alocação de suas máquinas virtuais em um ambiente de nuvem torna-se relevante para a confiabilidade do ambiente.

A *cloud computing* traz vantagens frente a integridade dos dados através da possibilidade de fornecer redundância de dados, *backups* automáticos ou criptografia de dados. Mas apesar das medidas de segurança existentes a comunicação e o acesso com os nós de uma rede dentro de um ambiente de *cloud* ainda pode fornecer vulnerabilidades ainda não exploradas por atacantes. Nos últimos anos tivemos o registro de alguns incidentes, como o ataque cibernético à empresa *SolarWinds*, que afetou vários órgãos governamentais dos Estados Unidos, como a *National Aeronautics and Space Administration (NASA)* e a *Federal Aviation Administration (FAA)*. O ataque foi derivado de uma falha de segurança na nuvem da *Microsoft* que permitiu extrair dados dos sistemas relacionados [34].

Em termos de análises técnicas podemos observar que de acordo com o relatório da empresa *Sophs* chamado de *State of Cloud Security*[1] de 2021 e com o relatório *Trustwave Global Security Report*[2] de 2020, pode-se constatar que 20% das organizações alvos da pesquisa sofreram algum tipo de ataque de *rasomware* na suas nuvens em 2020, e outras 30% tiveram dados sensíveis expostos por algum outro tipo de ataque. Além disso, as violações de dados em nuvens privadas superaram as violações constatadas em redes corporativas no ano de 2020.

Ao falar de integridade de dados, podemos nos referir também ao conceito de segurança que será bastante explorado ao decorrer do trabalho, segurança refere-se à proteção dos dados e informações armazenados e processados dentro de um ambiente computadorizado, sejam *clusters*, *private clouds*, *storages* e até mesmos dados trocados entre *hosts* e máquinas virtuais. A segurança torna-se um conceito mais amplo utilizando dentro de sistemas de informação ao considerar características como confidencialidade, a própria integridade, disponibilidade, autenticidade e não repúdio. Normalmente, ao se pensar em mecanismos de segurança comuns teríamos *firewalls*, criptografia, autenticação de usuários nos vários recursos acessados dentro de uma rede, bem como a detecção e prevenção de intrusões. Em contextos mais amplos, há de se falar em práticas de segurança, políticas gerais de acesso,

[1] https://www.qualys.com/docs/2021-cloud-security-report.pdf

[2] https://www.singtel.com/business/articles/2020-trustwave-global-security-report

gestão de senhas e atualização constantes de todos os softwares utilizados dentro do ambiente.

Há na literatura alguns trabalhos que definem formas de criação de máquinas virtuais a partir *flavors*, estabelecendo grupos de recursos específicos para criação de máquinas virtuais, onde não há rigor quanto aos detalhes específicos de cada implementação, bastando a necessidade do usuário estar contemplada nos limites estabelecidos no *flavor*. Existem outros trabalhos que permitem definir os detalhes dos recursos necessários como o número de núcleos do processador, tamanho do disco, tamanho da memória *RAM* e largura de banda [37]. Ao falar do estabelecimento prévio de critérios para alocação, considerar também aspectos de confiabilidade tornaria o processo de alocação ainda mais eficaz. A definição prévia dos requisitos pelo usuário ou sistema externo, proporcionaria maior controle dos critérios durante todo o ciclo de vida da máquina dentro do *cluster*.

Dada a alocação de uma máquina virtual dentro do *cluster* ela será monitorada com base nos requisitos estabelecidos. A possibilidade de monitoramento contínuo dos requisitos de confiabilidade das máquinas virtuais pelos seus usuários traz uma maior segurança. Utilizar os requisitos previamente estabelecidos como entrada para um processo de monitoramento contínuo dentro do *cluster* permitirá com que aquelas máquinas virtuais sempre permaneçam em *hosts* conforme a especificação inicial, garantindo a confiabilidade e segurança do ambiente.

Neste trabalho propomos um processo de alocação e migração de máquinas virtuais baseado em critérios de hardware, software e de confiabilidade, que é a principal contribuição deste trabalho. Todos os requisitos inicialmente estabelecidos serão armazenados em uma base de dados para verificação contínua. O processo de migração automática leva em consideração todos os requisitos solicitados e quando são identificadas inconsistências no estado atual dos *hosts*, todas as máquinas virtuais são migradas para um novo *host* que satisfaça os requisitos das máquinas virtuais. A seguir listamos as principais contribuições deste trabalho:

- Projeto e implementação de uma solução para consumo de dados gerados por uma ferramenta de atestação;
- Projeto e implementação de uma solução para automatização de tarefas no gerencia-

mento do *cluster*;

- Conhecimento mais aprofundado sobre integridade de sistemas computacionais;
- Auxílio no processo de alocação de máquinas virtuais;
- Auxílio do processo de migração de máquinas virtuais.

1.5 Organização do Trabalho

Os demais capítulos desse trabalho estão organizados da seguinte forma:

Capítulo 2 - Neste capítulo são apresentados o referencial teórico que serviu como base da pesquisa, os conceitos necessários ao entendimento do trabalho e os trabalhos relacionados com a pesquisa efetuada neste trabalho, destacando as semelhanças e diferença entre eles.

Capítulo 3 - Neste capítulo são apresentados o processo de decisão proposto neste trabalho, a definição arquitetural de cada componente, o modelo de dados utilizado, o modelo de ameaças, e uma descrição detalhada do processo de alocação e migração automática. Além dos detalhes do mecanismo de atestação utilizado para estabelecimento da confiabilidade dos nós. Também são apresentados os detalhes do componente de decisão utilizado durante todo o processo de alocação e migração.

Capítulo 4 - Neste capítulo é apresentada a avaliação experimental geral de desempenho e integridade, bem como a validação do processo de decisão proposto.

Capítulo 5 - Neste capítulo são apresentados a conclusão do trabalho, as considerações finais e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Virtualização

A virtualização tornou-se uma tecnologia bastante utilizada seja em ambientes corporativos ou acadêmicos. De forma simples, a virtualização permite com que determinados serviços originalmente vinculados a um tipo de hardware sejam fornecidos através de um software, permitindo com que aquele hardware original compartilhe os seus recursos com diversas outras máquinas. A virtualização tem sua origem datada na década de 1960, mas apenas nos anos 2000 passou a ser adotada amplamente. Foi originalmente pensada apenas para possibilitar acesso simultâneo de usuários a computadores que realizavam processamento em lote [18].

A virtualização permitiu uma customização de ambientes de hardware para ambientes de software, trazendo um novo conceito definido por [10] como "ambientes *softwarizados*". A virtualização possibilita vantagens do ponto de vista de segurança, como a possibilidade de isolamento e utilização de *sandboxs*, como também alguns serviços podem ser executados através de um gerenciador de virtualização, o que acarreta a necessidade de garantir a integridade do código e dos dados que são gerados por esses ambientes virtualizados.

De modo geral, um processo de virtualização ocorre com base no que se chama de *Virtual Machine Monitor (VMM)*, conforme pode ser observado na figura 2.1. O *VMM* é um componente que hospeda as máquinas virtuais e controla em baixo nível todo o processo de criação de VMs, assim como o compartilhamento de recursos entre todas as máquinas virtuais e entre elas e o hospedeiro. Além de gerenciar todos os recursos de hardware, o *VMM*

também é responsável por escalonar a prioridade de execução das máquinas virtuais [27].

O conceito de virtualização também leva em consideração níveis de execução dentro do processo, onde existe dois níveis básicos: o *nível do usuário* - que executa operações que não são privilegiadas, e o *nível de supervisor* - que executa operações que são privilegiadas, geralmente aquelas que envolvem a modificação de recursos físicos do hospedeiro. Nesse sentido, o *VMM* executa em *nível de supervisor* construindo uma camada entre as máquinas virtuais e o sistema de hardware da máquina hospedeira, enquanto que as máquinas virtuais executam no *nível de usuário*. Desse modo, quando é necessário que a máquina virtual realize uma operação privilegiada o *VMM* simula a execução do comando.

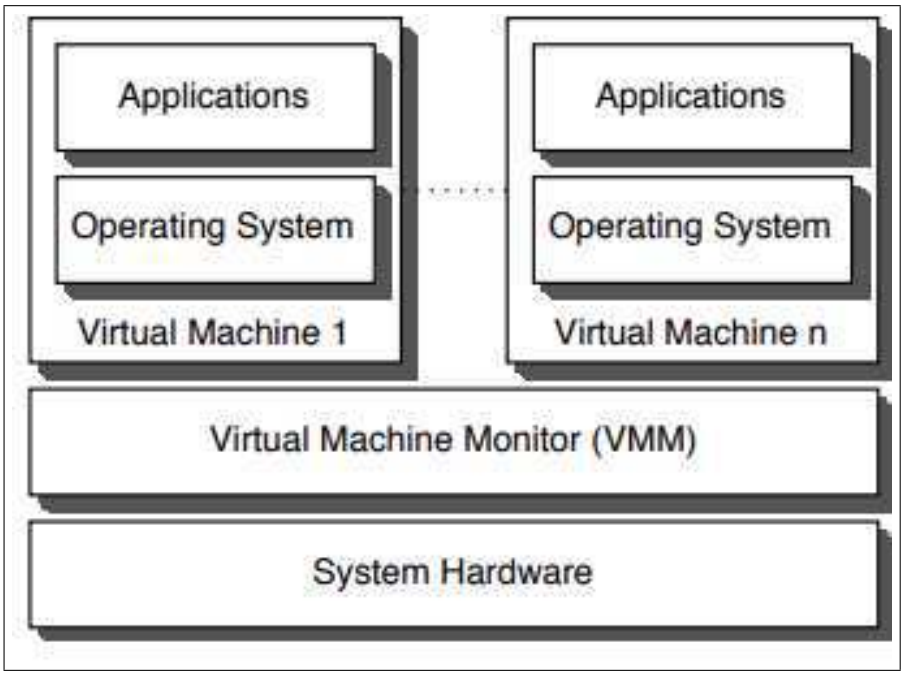


Figura 2.1: Arquitetura do Virtual Machine Monitor (VMM)

.

2.1.1 Virtualização Total ou *Hosted Virtualization*

A virtualização total tem o objetivo de fornecer ao sistema operacional requisitante uma réplica do hardware hospedeiro para executar diretamente sobre o *VMM* [27]. As instruções executadas pelo sistema operacional são repassadas para o *VMM* inspecioná-las e depois serem executadas no hardware, fazendo com que cada instrução solicitada seja verificada.

Além disso, a possibilidade de uso de um número elevado de máquinas virtuais rodando sobre *VMM* força-o a utilização de *drivers* cada vez mais genéricos, não permitindo em muitos casos a utilização da capacidade total do hardware.

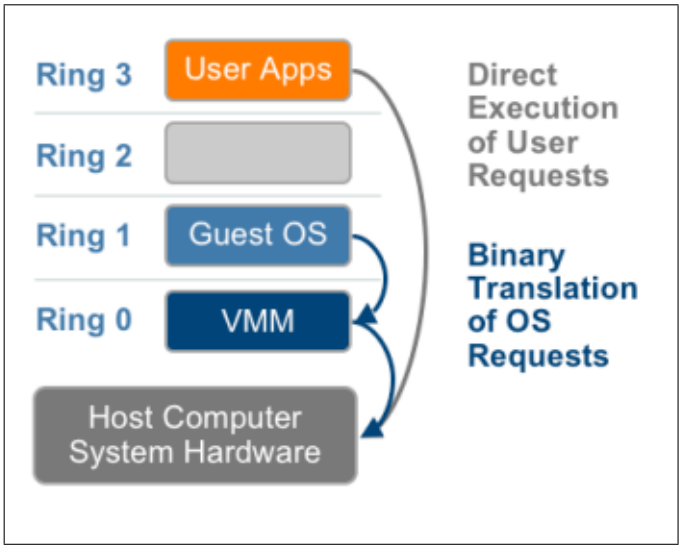


Figura 2.2: Virtualização Total.

2.1.2 Para-Virtualização ou *Bare-Metal Virtualization*

Neste modelo de virtualização, como pode-se observar na figura 2.3 o sistema operacional é modificado para chamar o *VMM* sempre que a máquina virtual executar uma instrução que possa alterar o estado do sistema. Essa característica otimiza o desempenho da virtualização, uma vez que o *VMM* não precisa verificar todas as instruções que são executadas. Além disso, o hardware é acessado diretamente por *drivers* da máquina virtual, permitindo a utilização total da capacidade do dispositivo.

2.1.3 Virtualização Assistida por Hardware

Na virtualização assistida por hardware, conforme apresentada na figura 2.4, há um privilégio das chamadas sensíveis que são automaticamente capturadas pelo *hypervisor* removendo a necessidade de tradução binária ou para-virtualização. Esse método de virtualização privilegia as instruções sensíveis que são automaticamente capturadas pelo *hypervisor*, rodando de

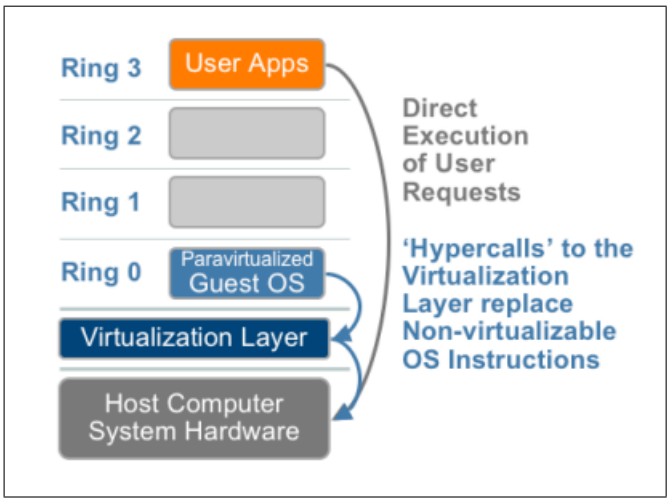


Figura 2.3: Processo de Para-virtualização.

maneira nativa e removendo a necessidade de tradução binária das instruções ou a utilização da para-virtualização.

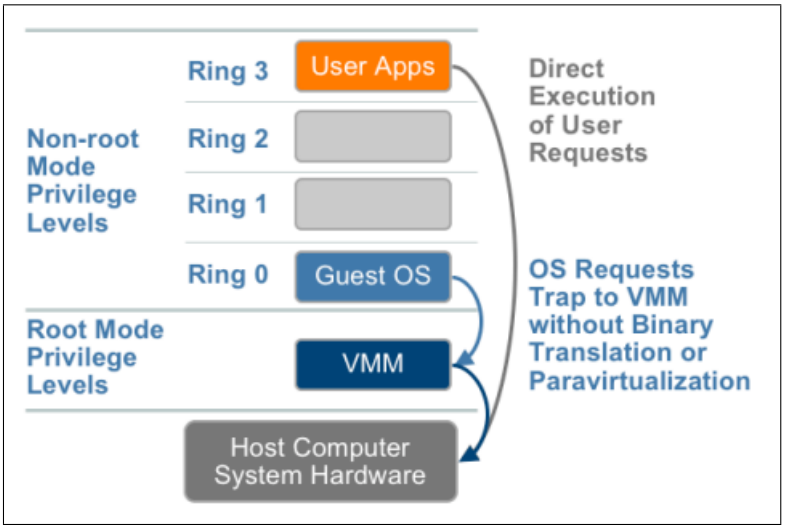


Figura 2.4: Arquitetura da Virtualização Assistida por Hardware.

Virtual Machine Monitor (VMM)

No final dos anos 1960, o *Virtual Machine Monitor (VMM)* iniciou como uma camada de abstração que particionava o hardware em uma ou mais máquinas virtuais [29]. Nas décadas seguintes com a chegada dos sistemas operacionais multitarefas, a tecnologia *VMM* ganhou

um maior espaço na academia e na indústria. As pesquisas para superar as limitações de hardware para implantação de máquinas virtuais foram intensificadas e representaram um grande salto na evolução dos monitores de máquinas virtuais. A figura 2.1 detalha a arquitetura do *VMM*, demonstrando que a sua utilização desacopla o software do hardware formando uma camada intermediária.

Xen

É um projeto focado no avanço da virtualização em uma variedade de aplicações *open source* e comerciais, incluindo virtualização de servidores, *Infrastructure As a Service (IAAS)*, aplicações de segurança, aplicações embarcadas e de hardware. A tecnologia do *Xen*[1] está disponível para o *Kernel Linux* projetado para consolidar múltiplos sistemas operacionais para executarem em um único servidor, normalizando o acesso ao hardware, isolando aplicativos comprometidos e permitindo a migração de instâncias de sistemas operacionais que estão em execução para outro servidor físico.

2.1.4 Virtualização a Nível de Kernel

KVM

O *Kernel Virtual Machine (KVM)* é uma tecnologia de virtualização *open source* presente no *Kernel* do Linux desde a versão 2.6.20, tornando-s uma vantagem para o processo de virtualização no *Linux*. O *KVM* converte o sistema operacional em um hipervisor *bare-metal*, e como faz parte do *kernel* do Linux utiliza nativamente os componentes necessários à virtualização como gerenciador de memória, agendador de processos, entrada e saída (E/S), *drivers* de dispositivo, gerenciador de segurança, entre outros componentes [19]. Nesse tipo de virtualização, cada máquina virtual é implementada como um processo comum do Linux e agendada diretamente pelo programador padrão do sistema usando hardware virtual dedicado.

Implicitamente, o *KVM* utiliza o *QEMU* - software de emulação anterior e independente do *KVM* - para criar a máquina virtual. Como o *QEMU* trabalha a nível de software acabada sofrendo limitação de desempenho em decorrência disso. A interação entre os dois

[1] https://xenproject.org/

componentes acontece através da solicitação e execução de recursos. O *kernel Linux* cria um dispositivo em */dev/kvm* que permite ao *QEMU* realizar pedidos ao hipervisor [12]. Para criar uma máquina virtual, o *QEMU* faz o pedido ao hipervisor através de uma instrução para a criação, ao receber a solicitação o *KVM* muda o modo de execução de usuário para o modo de operação de supervisor, e durante o processo de análise de instruções, ao chegar uma requisição de operação sensível o *KVM* realiza a mudança de modo superusuário para o modo de operação não-privilegiado.

2.1.5 Emulação, simulação e virtualização

O conceito de emulação, do ponto de vista computacional, é um modelo matemático que representa características chaves de um determinado domínio para o entendimento de seu comportamento ou funcionamento. O emulador é capaz de executar o mesmo ambiente do hardware hospedeiro no hardware virtualizado, permitindo com que máquinas virtuais tenham acesso a uma espécie de cópia do ambiente original. O simulador, por outro lado, é um programa que implementa um modelo de sistema a partir de determinados parâmetros de entrada e condições iniciais de contorno [6].

O termo simulador tem sido empregado em diversas áreas dentro da física, química, matemática e biologia para simular modelos que refletem testes e representações fenômenos naturais. No aspecto da virtualização, podemos defini-la como uma metodologia ou técnica em que os recursos computacionais físicos são divididos e compartilhados em múltiplos ambientes de execução, permitindo que em um mesmo ambiente hospedeiro seja possível a existência de múltiplas máquinas virtuais [24].

2.1.6 Ferramentas de Virtualização

VMware

- **VMware ESX e ESXi:** É um *hypervisor baremetal* instalado diretamente no servidor físico que abstrai os recursos de processador, memória, armazenamento e rede de várias máquinas virtuais. A principal vantagem de utilização desse tipo de hipervisor é a ausência da necessidade de um sistema operacional de uso comum, devido a característica de poder ser instalado diretamente no hardware.

- **VMware vSphere:** É uma plataforma de virtualização da *VMware* que transforma os centros de dados em uma infraestrutura de computação agregada com gerenciamento de todos os nós compartilhando recursos.

- **vCenter Server:** É a ferramenta da *VMware* que gerencia vários *hosts* conectados a uma rede e realiza o gerenciamento do compartilhamento de recursos do *host* entre máquinas virtuais alocadas. Na Figura 2.5 podemos visualizar a arquitetura do *VMware vSphere*, através dos componentes *ESXi* e *vCenter* é possível a construção de um *cluster* com vários *hosts ESXi* instalados diretamente no hardware e compartilhando seus recursos. Na infraestrutura, tem-se um servidor *vCenter* e opcionalmente mais de um *vSphere* gerenciando a infraestrutura. Cada *ESXi* pode abrigar uma ou mais máquinas virtuais, assim como o *vCenter* pode ter um ou mais *hosts ESXi*. Cada *vSphere* gerencia um *cluster* e através da *user interface*, *APIs* ou *SDKs* é possível o gerenciamento de todos os nós de cada *cluster* [36].

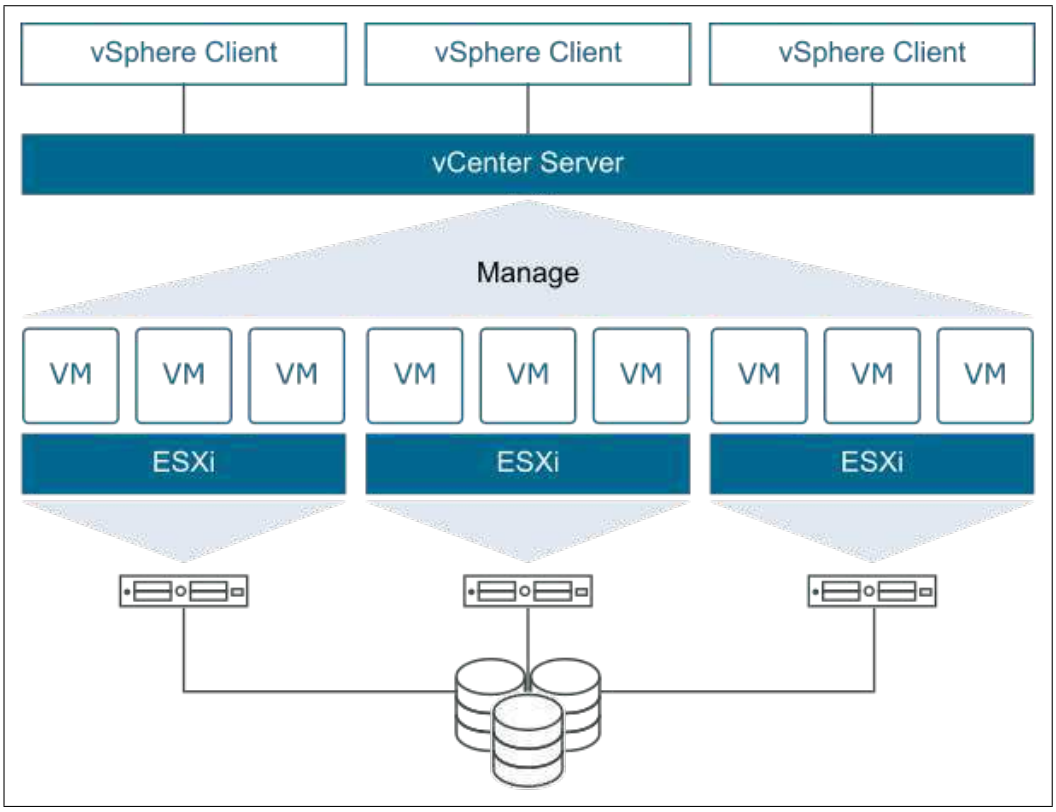


Figura 2.5: Arquitetura do VMware vSphere

- **VMware Workstation:** É mais uma ferramenta de virtualização da *VMware* para

ambientes *desktop*, conta com a versão *Player* e a versão *Pro*. A *Player*[2] é a versão gratuita destinada ao público em geral que deseja iniciar em ambientes virtualizados, com acesso aos recursos comuns como criação de máquinas virtuais através de uma interface intuitiva e suporte para versões de *Windows Server*. A versão *Workstation Pro*[3], é a versão paga e completa dos virtualizadores *desktop* da *VMware*, conseguindo operar várias máquinas virtuais ao mesmo tempo, além de permitir a conexão com o *vSphere* e o *ESXi - hypervisor* padrão da *VMware*. Além disso, conta com as funções de clone, *snapshots* e compartilhamento de máquinas virtuais.

- **VMwareFusion**[4]**:** É um *hypervisor* desenvolvido para computadores *Macintosh* e que permite a virtualização de outros sistemas operacionais como *Windows* e *Linux*, rodando de maneira nativa ao sistema operacional *OS X* da *Apple*.

VirtualBox

O *VirtualBox*[5] é um software de virtualização de plataforma cruzada que permite aos desenvolvedores executar vários sistemas operacionais em um único dispositivo [32]. O *VBox* roda sobre um sistema operacional já instalado em um hospedeiro *Windows*, *Linux* ou *MAC*, e permite uma utilização facilitada por uma *user interface* em que é possível abstrair configurações mais baixo nível em sistemas *checkbox* de seleção. O *VBox* roda sobre um hardware virtual, assim qualquer modificação realizada no sistema operacional deste hardware não interfere no funcionamento do hospedeiro. A virtualização usada por essa técnica é executada apenas no modo usuário, o que significa que qualquer tentativa de execução de uma instrução sensível será ignorada pelo *VMM*. A vantagem de utilização do *VBox* é a abstração realizada de diversas configurações, além de possuir uma ampla variedade de opções de sistemas operacionais para instalação.

[2]https://www.vmware.com/products/workstation-player.html

[3]https://www.vmware.com/br/products/workstation-pro.html

[4]https://www.vmware.com/files/br/pdf/products/08Q3_VM_FUSION2_DS_PTB_A4_R1.pdf

[5]https://www.virtualbox.org/

Microsoft

- Hyper-V[6]: É o software de virtualização da *Microsoft* semelhante ao *VirtualBox*. Este software roda sobre um hardware virtual específico para cada máquina virtual criada. Apesar de só está disponíveis nas versões *Windows 10 Pro* e versões do *Windows Server*, este software permite a adição de discos rígidos virtuais, comutadores virtuais e outros dispositivos virtuais como TPM. O *Hyper-V* também oferece um conjunto de funcionalidades diferentes a depender da versão do hardware hospedeiro em que está sendo utilizado, tendo sua versão mais robusta disponibilidade no *Windows Server*.

2.2 Integridade em Sistemas Computacionais

2.2.1 *Trusted Platform Module (TPM)*

O *Trusted Platform Module (TPM)* é um chip acoplado a placa-mãe de um computador. Esses chips são avaliados pela entidade fabricante permitindo com que sejam certificados para utilização da máquina e garantindo a autenticidade do chip. Para o funcionamento do TPM, o hardware e o *firmware* do sistema devem estar integrados ao TPM para o envio de comandos e a espera de respostas.

A especificação do TPM é mantida pelo *Trusted Computing Group (TCG)* que é uma organização sem fins lucrativos com o objetivo de desenvolver, definir e promover padrões globais de confiança baseada em hardware para plataformas de computação confiáveis interoperáveis. A utilização do chip TPM físico fornece uma confiabilidade que não é possível exclusivamente através de software, principalmente quando se quer garantir um ambiente confiável desde o processo de *boot* do sistema operacional.

O funcionamento do TPM está relacionado com o processamento de chaves criptográficas realizando operações de geração, armazenamento e limitação do uso de chaves. Dentre as vantagens de utilização do TPM estão a possibilidade do gerenciamento de chaves criptográficas, autenticação de dispositivos usando a chave *RSA* exclusiva do TPM e a garantia de integridade da plataforma a partir da utilização de suas chaves.

É possível a virtualização do chip TPM, chamado de *Virtual Trusted Platform Module*

[6]https://learn.microsoft.com/pt-br/virtualization/hyper-v-on-windows/about/

(vTPM), que simula o comportamento do chip TPM real para garantir a integridade de ambientes virtualizados que não possuem o chip físico. A utilização do vTPM trouxe diversas vantagens para os ambientes virtualizados, permitindo com que todos os nós dentro de um *cluster* virtualizado possam ter a mesma garantia de integridade de uma máquina com chip físico.

2.2.2 *Integrity Measurement Architecture (IMA)*

O *IMA* é subsistema *Linux* que introduz "ganchos"no *kernel* para suportar a criação e coleta de *hashes* de arquivos quando eles forem abertos. O arquivo do *IMA* é responsável pela verificação de integridade de *paths* executados dentro do sistema operacional e foi adicionado ao *Linux* a partir a versão de *kernel* 2.6.3 [31]. A arquitetura geral do *IMA* consiste em dois componentes: (1) *IMA* - responsável por coletar os *hashes* dos arquivos e armazená-los em memória de *Kernel (PCRs)*, e também por permitir a verificação dos valores medidos por entidade atestadores locais ou externas; (2) *Extended Verification Module (EVM)* responsável por detectar qualquer alteração off-line nos atributos de seguranças estendidos.

O *Trusted Computing Group (TCG)* mantém uma especificação detalhada e atualizada à medida que novas funcionalidades são incorporada na linguagem de especificação. O *TCG* especifica quais eventos serão coletados, como serão armazenados e qual a sintaxe de armazenamento desses dados nos arquivos [25]. Isso permite com que ocorra uma padronização na forma de recuperação dos arquivos e também na forma de implementação de *APIs* de comunicação.

2.2.3 *Windows Boot Configuration Log (WBCL)*

O *WBCL* faz parte de um conjunto de ferramentas da *Microsoft* para segurança chamado *TCGLogTools*. O arquivo representa um l*log* de eventos medidos de *boot* e para que seja coletado é necessário está rodando em uma versão mínima de *Windows* 8 com o TPM habilitado. O *WBCL* também segue a especificação do *TCG* quanto a nomenclatura de eventos. Esse arquivo de *log* registra todos os estágios do processo de inicialização do sistema operacional que são usados para fins de atestação de integridade do dispositivo, podendo ser usado de forma interna pelo próprio sistema operacional ou por uma entidade de atestação externa.

2.2.4 *Remote Attestation (RA)*

A *Remote Attestation (RA)* ou atestação remota é um mecanismo que define a comunicação entre um sistema interno e um sistema externo, a fim de que esse sistema externo realize uma atestação de integridade ou confiabilidade do sistema interno [4]. Na literatura e na indústria há algumas possibilidade de garantir a atestação remota de sistemas a partir de diferentes perspectivas e objetivos.

2.2.5 Modelo de atestação remota

A Figura 2.6 traz um modelo de atestação remota comumente implementado por diferentes tecnologias. Para um sistema ser atestado é necessário inicialmente obter a chave pública do *TPM Vendor* que gera uma *Endorsement Key* and OEM gerada a partir da *Attestation Key* utilizando os certificados que podem ser obtidas a partir de uma localização pré-definida da memória *NonVolatile*, também pode ser obtida do fabricante. A partir dessa verificação, as chaves são armazenadas em uma base de dados local integrada com a infraestrutura de geração de chaves.

No processo de atestação também pode ser armazenados a *policy* que define o estado inicial esperado do dispositivo, e um *digest* que resume o estado inicial recebido do sistema. Os valores de referência do *boot* do software precisa ser medido durante o processo de inicialização armazenando valores de *bootloader*, sistema operacional, *drivers* e os espaços de aplicações de usuário. As medições são armazenadas em uma base local para serem usadas durante o processo de atestação e determinar se o nó encontra-se em um estado de integridade.

vSphere

O *vSphere* é a plataforma de virtualização da *VMware*, nela tem-se o *vCenter* que é um nó gerenciador de todo o *cluster*. A atestação remota no *vSphere* é realizada através do *vCenter* que constantemente monitora o *health status* dos *hosts*. Essa verificação parte da análise do status de integridade fornecido pelo chip TPM do *host* físico. Por meio dele é possível verificar todos os eventos executados durante o processo de inicialização e durante a execução do hipervisor, caso tenha sido detectado qualquer alteração não reconhecida nos

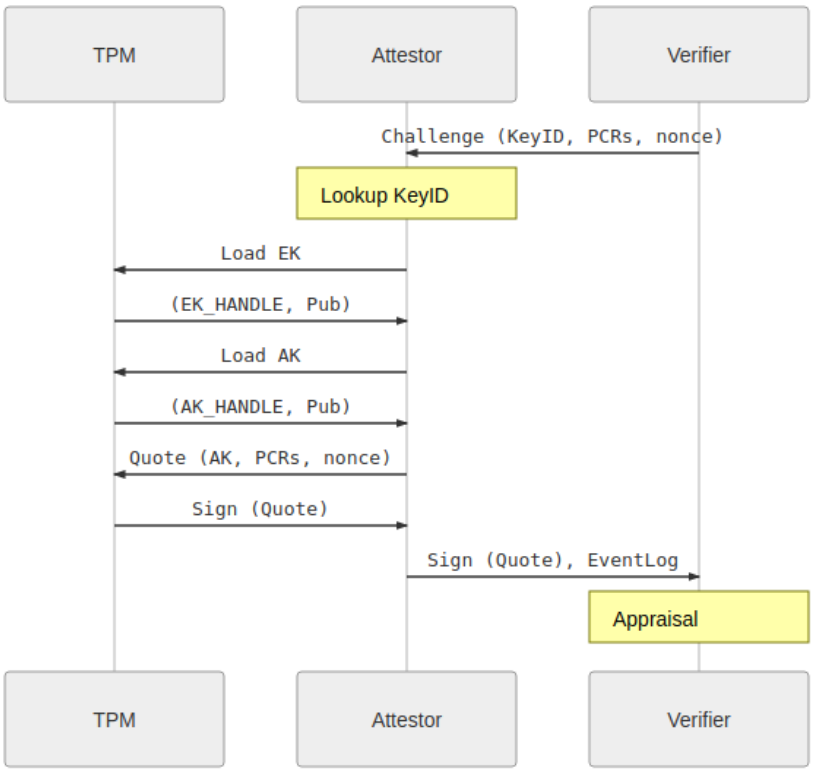


Figura 2.6: Modelo de Atestação Remota [8].

eventos, o *vCenter* emite um alarme de TPM para o *vSphere* que passa a não considerar mais aquele *host* como seguro.

SPIFFE SPIRE

O *SPIRE*[7] é uma implementação para ser usada por meio de *APIs SPIFFE* que executam a atestação do nó a partir de verificação de cargas de trabalho. O objetivo desse tipo de atestação remota é garantir a continuidade dos serviços dentro de uma infraestrutura, assim pode-se definir um conjunto de condições para serem verificados no processo de atestação realizado pela ferramenta.

A arquitetura do SPIRE pode ser observada na Figura 2.7, e é composta de uma entidade servidora e um ou mais agentes. O servidor age como uma autoridade de assinatura para identidades emitidas para um conjunto de cargas de trabalho por meio de agentes [17]. Como o *SPIRE* atua com cargas de trabalho, também é possível manter o registro de cargas de

[7]https://spiffe.io/docs/latest/spire-about/spire-concepts/

trabalho existentes e as condições que devem ser verificadas para que as identidades sejam emitidas. Os agentes são expostos por uma *API SPIFFE* que deve ser instalada em cada nó onde existe uma carga de trabalha em execução.

O *SPIRE Server* é responsável por gerenciar e identificar e emitir todas as identidades no domínio do SPIFFE previamente configurado, armazenar das entradas de registro e as chaves de assinatura. O *server* utiliza atestação de nó para autenticar os *agents* automaticamente e criar SVIDs para cargas de trabalho quando solicitado por um agente autenticado.

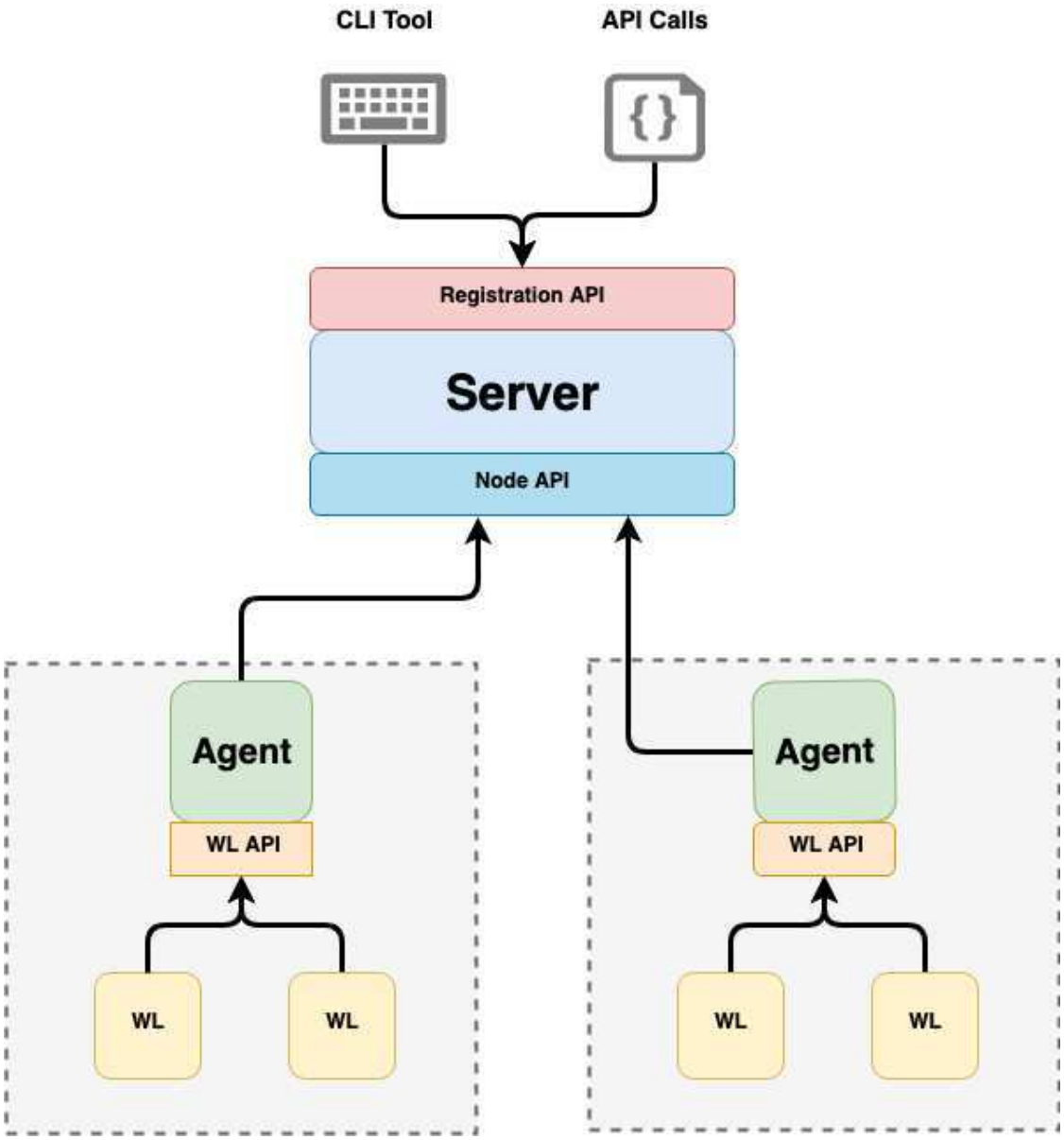


Figura 2.7: Arquitetura do SPIFFE SPIRE [17].

O comportamento do servidor é determinado através de um conjunto de *plugins* adicionados para fornecerem funcionalidades adicionais para a arquitetura. Entre os *plugins* disponíveis é possível encontrar:

- **Node attestor**: responsável por verificar a identidade do nó no qual o agente está sendo executado.
- **Node resolver**: *plugin* de resolução de nó que expande o conjunto de seletores que o servidor pode usar para identificar o nó e verificar propriedades adicionais que podem podem ser aplicados sobre o nó.
- **Datastore**: Todos os *plugins* relacionados ao armazenamento, consulta e atualização de entradas de registro, nós que realizarão a atestação, nós que serão atestados e quais os seletores serão aplicados em cada nó. Existem também os *plugins* relacionados às variações de bases de dados como *MySQL*, *SQLite 3* ou *PostgresSQL*.
- **Key manager**: *plugins* que controlam como o servidor armazena as chaves privadas usadas para assinar *X.509-SVIDs* e *JWT-SVIDs*.
- **Upstream authority**: *plugins* de autoridade, que atua como sua própria autoridade de certificação, mas é possível usar um *plug-in* de autoridade *upstream* para usar uma CA diferente de um sistema *PKI* diferente.

O *SPIRE Agent* executa em cada nó onde uma carga de trabalho é identificada. Os principais *plugins* do *agent* incluem *node attestors*, *workload attestor* e *key managers plugins*. No fluxo de execução do *agent* temos as seguintes características:

- Solicita as *SVIDs* do servidor e armazena em cache até que uma carga de trabalho solicite seu *SVID*;
- Expõe a *API SPIFFE Workload* para cargas de trabalho no nó;
- Atesta a identidade das cargas de trabalho.

O *SPIFFE SPIRE* traz uma série de componentes extras e *plugins* relacionados a cada componente. O fluxo de alguns componentes são semelhantes à de outras ferramentas de

atestação e apesar do foco da ferramenta não ser apenas de um processo de atestação, apresenta uma série de mecanismos semelhantes como um fluxo de registro de novos nó e um processo de atestação baseado na autenticação inicial do nó solicitante.

Keylime

O *Keylime*[8] é um projeto que fornece uma solução altamente escalável de atestação remota do processo de *boot* e para medição de integridade em tempo real [16]. O projeto foi inicialmente desenvolvido pelo time de pesquisa em segurança do *MIT'S Lincoln Laboratory* e atualmente é mantido pela comunidade *Keylime* com o a apoio da *Cloud Native Computing Foundation (CNCF)*.

A arquitetura do *Keylime* pode ser observada na Figura 2.8, e consiste de um agente, dois componentes servidores - o *verifier* e o *registrar* e uma ferramenta de linha de comando para interagir com todo o *cluster* chamada de *tenant*. O componente *agent* instalado em cada sistema operacional que fará parte como um nó do *cluster*, é responsável por se comunicar com o TPM para recuperar a *Attestation Key (AK)*, geração dos quotes e coleta do IMA para verificação de integridade. No contexto do *agent*, ele funcionará como interface de comunicação entre o sistema operacional do nó e o servidor do *Keylime* responsável por realizar a coleta de todos os dados necessários para o funcionamento correto do monitoramento do nó na entidade servidora.

O componente *Registrar* fica do lado servidor e é responsável por receber os registros dos *agents* e gerenciar todo o processo de recebimento dos dados coletados do lado cliente. No processo, o *Registrar* recupera o *UUID* do *agent*, a *Endorsement Key (EK)* pública, o certificado da *EK* e a *Attestation Key (AK)* pública do agente e verifica se os valores correspondem a partir da validação das chaves do TPM. Uma vez que o TPM do nó é validado a partir da atestação das chaves, o nó é automaticamente adicionado ao *cluster* e passa a ser monitorado em tempo real. Vale ressaltar que diferente de outros mecanismos de atestação, a validação do TPM do nó é realizada apenas no primeiro contato do *agent* com o *registrar*. O monitoramento em tempo real subsequente é apenas em relação a integridade do arquivo de medições.

O componente *Verifier* implementa a atual atestação do *Keylime* e se comunica direta-

[8] https://keylime.readthedocs.io/en/latest/

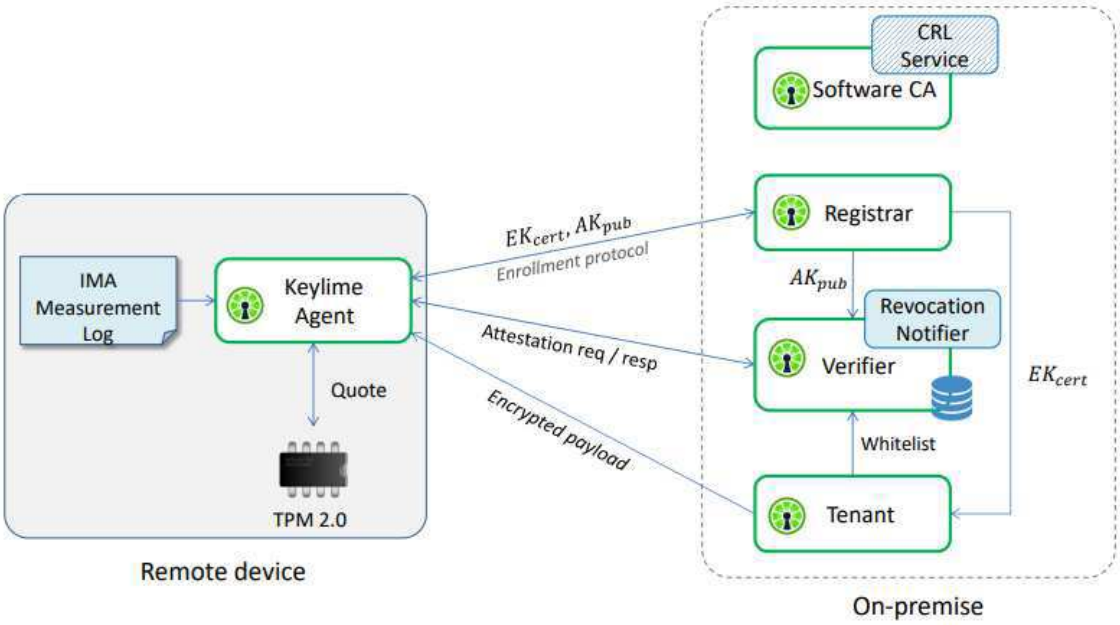


Figura 2.8: Arquitetura do Keylime [33].

mente com o *Registrar* e com o *agent* de cada nó cliente. Além de realizar a atestação do arquivo de medição, este componente é responsável por enviar mensagens de revocação em caso dos *agents* deixarem o estado confiável. Uma vez que um *agent* é registrado para atestação, o *verifier* continuamente envia requisições para os *agents* para coleta dos dados de atestação.

O *tenant* é o último componente da arquitetura e funciona como uma ferramenta de linha de comando para gerenciar todos os *agents*, com ele é possível adicionar e remover *agents*, validação da EK e recuperar os *status* de um *agent*. O *Keylime* permite o monitoramento em tempo real e a definição de regras para cada nó em particular. Possui uma implementação inicial em *Python* permitindo uma boa interoperabilidade entre ambientes e atualmente passa por uma migração da sua base para a linguagem *Rust*[9].

2.3 Tomada de Decisão

A tomada de decisão ganhou relevância frente a era do *Big Data*, as vantagens obtidas com o gerenciamento de *big data* podem ser aplicadas por meios tecnológicos, negócios digitais

[9]https://www.rust-lang.org/pt-BR

ou on-line [15]. Uma vez que os dados representam um *snapshot* da organização naquele instante de tempo, a reconstrução do percurso de um sistema de informação possibilita identificar pontos de falhas e inconsistências no ambiente.

Dentro de um negócio, a tomada de decisão pode ser utilizada de forma corretiva ou preventiva quando há um perigo iminente nos seus sistemas [2]. Desse modo, dados temporais coletados dentro de um contexto organizacional podem ser utilizados para avaliar comportamentos passados e prever comportamentos futuros, sejam eles bons ou ruins, neste caso, faz-se necessário a utilização de medidas preventivas, a partir da identificação de pontos críticos que podem gerar falhas futuras.

2.4 Trabalhos Relacionados

Esta seção apresenta um levantamento de trabalhos relacionados detalhando trabalhos que utilizam métricas de segurança para a tomada de decisão, além de abordar a importância da análise eficiente de dados para agregar valor aos dados históricos gerados por uma organização. Nesse contexto, também abordamos trabalhos relacionados a integridade de sistemas computacionais, buscando detalhar os aspectos técnicos e teóricos.

2.4.1 *Remote Attestation* na Garantia da Integridade

Para George Coker et al. [7], a *Remote Attestation* é um conjunto de procedimentos que permitem a uma parte confiável tomar decisões baseada em evidências fornecidas por entidades dentro do seu sistema. A literatura traz diversos trabalhos relacionados ao tema em questão, a garantia de integridade é um tópico em alta desde que a segurança passou a ser fator chave nas comunicações. No aspecto corporativo e empresarial, com a adoção de *clouds privates*, torna-se necessário pensar novos aspectos de proteção e segurança para fornecer autenticidade e confiabilidade dentro da *cloud*. A utilização do chip TPM torna-se um fator primordial para garantir a base de segurança dessa comunicação.

O trabalho de Haonan Sun etal. [35] desenvolve um método de atestação remota e certificação utilizando-se do eTPM, que é uma variação de implementação do TPM físico, que é utilizada como base o ambiente de virtualização do Xen. O eTPM é utilizado para atender aos requisitos de autenticação de identidade entre departamentos internos de uma *cloud*

corporativa.

A arquitetura do eTPM utilizada como base para o desenvolvimento do trabalho faz uso da chave privada *Attestation Identity Key (AIK)* do TPM para assinar a própria instância. Quando o usuário requisita logar em uma máquina, o servidor na *cloud* permite apenas aquelas máquinas cuja chave privada do TPM seja correspondente com o chave pública fornecido pelo cliente. Uma vez que as partes são autenticadas, o chip TPM do cliente requisitante é validado e o sistema pode realizar o *boot* de forma segura. Para a verificação de integridade *runtime*, alternativamente à utilização do vTPM, o trabalho faz uso da certificação da chave privada do eTPM chamada de *eAIK*. Através de uma associação realizada entre a *AIK* do TPM físico de uma máquina do departamento e da *eAIK* pode-se realizar uma associação completa e certificar a *eAIK*.

Assim, o trabalho consegue garantir a integridade de *boot* e de *runtime* através da utilização de um método de autenticação de identidade que leva em consideração o processo de *boot* de usuários de um *cluster* privado, uma entidade que recebe as chamadas para validação e o *runtime* é garantido através das assinaturas dos registros realizados na máquina, permitindo com que apenas a mesma entidade possa realizar a comunicação para ter acesso aos dados gerados.

Nesse sentido, a atestação remota permite à uma entidade previamente confiável tomar decisões confiáveis, diferenciais e dependentes do contexto em que está inserida. Sarah C. Helble et al. [20] realiza um estudo de vários mecanismos de atestação destacando a importância de um mecanismo dinâmico que seja capaz de lidar com as variações de ambientes, através da troca de diversos tipos de formatos de mensagens.

O trabalho é implementado através da perspectiva de existências de vários casos de uso, em que a proposta pode ser implantada. Para alcançar esse objetivo é utilizada uma *attestation protocol language* chamada de *Copland*[10], projetada para especificação formal e desenvolvimento de ferramentas para atestação remota. Sistemas como *Maat* e *Haskell Attestation Manager* foram construídos e empiricamente avaliados [20] utilizando a proposta apresentada no trabalho.

Nos trabalhos apresentados, a utilização da atestação remota desempenha um papel de fundamental importância para garantir a confiabilidade na interação entre os diferentes com-

[10]https://ku-sldg.github.io/copland/

ponentes de um sistema, seja diretamente na comunicação entre nós em uma rede ou seja entre entidades clientes e servidores de um ambiente de *cloud enterprise*.

2.4.2 Sistemas de apoio à tomada de decisão

Sistema de apoio à tomada de decisão ou sistemas de apoio à decisão (SAD) ajudam administradores a tomar decisões eficientes diante de cenários complexos no contexto do seu negócio. A necessidade de realização de tarefas cada vez mais rápidas aliado à necessidade de obter a satisfação de clientes obrigam os gestores de grandes empresas a se tornarem cada vez mais competitivos diante do cenário vivenciado.

Dentro desse cenário dinâmico Mahsa [13] desenvolve um *framework* baseado em sistemas de apoio à decisão e *self-protection software (SPS)*. A interação entre os componentes do *framework* ocorre através da implementação de um modelo baseado na teoria dos jogos de *game two-player*. O cenário é construído com um atacante e um sistema sendo atacado, à medida em que o atacante tenta realizar determinadas atividades, o sistema dinamicamente tenta selecionar uma contramedida adequada. O trabalho destaca os pontos positivos da utilização desse tipo de abordagem no que tange à dinamicidade com que uma contramedida pode ser selecionada e alinhada com os objetivos específicos do sistema de software.

Kamilia [1] desenvolve um trabalho relacionado ao conceito de *trust-based desicion making* através de *agents self-adaptive* com o meio relacionado. O trabalho é construído a partir da perspectiva da diminuição da confiabilidade na execução de uma tarefa quando múltiplos agentes estão envolvidos. Desse modo, a criação de *agents* através de um cenário de confiabilidade, permite com que a troca de mensagens seja baseada em tomadas de decisões a partir dos dados de interação com o meio que foram coletados.

2.4.3 Escalonamento

Escalonamento de sistemas, escalonamento de tarefas ou escalonamento de processos é um conceito relacionado à sistemas operacionais e computação distribuída. Relaciona-se à maneira como o sistema realiza o gerenciamento e a priorização de execução de várias tarefas ou processos utilizando-se do conceito de concorrência, onde um ou mais processos estão concorrendo simultaneamente para utilização de um determinado conjunto de recursos como

tempo de CPU, memória, dispositivos de entrada/saída, entre outros.

O conceito de escalonar um sistema está relacionado à decidi a ordem de execução das tarefas, levando em a prioridade dos processos e a concorrência na utilização dos recursos da máquina. Utilizar técnicas de escalonamento em um ambiente relaciona-se ao processo de maximização de desempenho do ambiente, buscando otimizar os tempos de retornos e os tempos de duração e consumo dos demais recursos.

O objetivo principal é otimizar o desempenho do sistema, para isso existem diversos algoritmos que podem ser utilizados a depender de particularidades específicas que se deseja no processo de execução e alocação de recursos. Os principais algoritmos utilizados estão destacados abaixo:

- **Escalonamento por prioridade**: cada tarefa pode possuir uma prioridade estática ou uma prioridade dinâmica que muda durante o processo de execução.
- **Escalonamento *round-robin***: cada tarefa recebe um intervalo de tempo de CPU igual.
- **Escalonamento por loteria**: as tarefas recebem bilhetes com base na sua prioridade.
- **Escalonamento por tempo real**: as tarefas têm requisitos rígidos de tempo.

O trabalho de Santos Júnior [23] apresenta um processo de escalonamento em sistemas de tempo real multiprocessados com baixo custo de implementação. Trazendo a contextualização do avanço da utilização de plataformas de múltiplas unidades de processamento com a necessidade de escalonar tarefas levando em consideração aspectos de paralelismo para garantia de restrições temporais de STR.

O desafio nesse tipo de abordagem é escalonar *n* tarefas em um sistema de tempo real executando em uma plataforma composto de diversos processadores. O trabalho propõe um método de escalonamento chamado de *Hime (HIghest-priority Migration managed by EDF)* onde a maioria das tarefas são executadas em um único processador e o número de processos migratórios é muito menor. Durante o processo de avaliação, foi constatado que o HIME é capaz de lidar com sistemas em até 95% de utilização apresentado bom desempenho e manuteção da integridade dos dados.

Capítulo 3

Abordagem Proposta

3.1 Contextualização e Descrição da Ferramenta

A abordagem proposta é uma ferramenta para gerenciamento e otimização dos processos de alocação e migração de máquinas virtuais dentro de uma infraestrutura de um *cluster*, podendo ser usada tanto em soluções de computação na nuvem quanto com hipervisores "convencionais". Para tanto, a abordagem proposta se baseia em dados gerados por uma entidade externa capaz de fornecer dados da atestação do estado do ambiente para a definição do grau de confiabilidade das máquinas do *cluster*. Entre as ferramentas possíveis para realização de atestação em ambiente de *cloud* tem-se, por exemplo, o vSphere[1] para *clusters ESXi*, o *Keylime*[2] para *cluster* com máquinas virtuais Linux e o *Spire*[3] para *clusters* Linux, MacOS ou *deploy* a partir de imagens *Docker*. Os dados também podem ser coletados por *agents* externos implementados de forma independente, desde que sejam construídos com base na confiabilidade e segurança fornecida a partir da ancoragem com o chip TPM ou qualquer outro mecanismo de atestação ancorado em hardware.

A ferramenta se comunica diretamente com a entidade que realiza o gerenciamento do *cluster* para requisitar informações de hardware relacionados aos *hosts* e as máquinas virtuais presentes no *cluster*. Além disso, a ferramenta também realiza o consumo do serviço de atestação para obter os indícios de confiabilidade dos hosts. Para que todas as etapas

[1] https://www.vmware.com/br/products/vsphere.html

[2] https://keylime.dev/

[3] https://spiffe.io/

sejam realizadas corretamente é necessário que as máquinas do *cluster* sejam monitoradas e continuamente atestadas para que a abordagem proposta possa considerar os aspectos de confiabilidade especificados pelos usuários.

Uma vez que a base de dados é alimentada por uma entidade atestadora, a ferramenta poderá realizar o gerenciamento de *hosts* e máquinas virtuais a partir de determinados requisitos definidos pelo usuário no momento de criação das máquinas virtuais. Dentre os requisitos possíveis, pode-se destacar os requisitos de software e hardware presentes em toda infraestrutura de *cloud*, e o requisito de confiabilidade, possibilitado através da utilização da ferramenta, onde será possível definir parâmetros específicos, como *encryption policy* ou presença do chip TPM/vTPM.

A partir da base de dados será possível consumir os dados gerados pelo mecanismo de atestação e utilizá-los para basear a decisão do processo de alocação. Será possível também utilizar os dados gerados para basear as decisões de migração automática realizada pela ferramenta proposta durante o processo de monitoramento do *cluster*. Caso uma máquina virtual não se encontre mais em um estado de segurança e integridade esperado, e esse estado seja irreversível, com base em uma *policy* que poderá ser definida pelo usuário no momento da alocação, pode-se continuar a execução, proceder a retirada do *host* onde a máquina virtual se encontrava ou a remoção da máquina virtual do *cluster*.

3.1.1 Visão Geral da Arquitetura

A arquitetura da ferramenta pode ser observada na Figura 3.1 que apresenta todos os componentes internos e externos da ferramenta proposta. O componente nomeado de ***Tool Manager*** é o componente geral e principal, nele encontram-se a interface de comunicação, que envia requisições de alocação ou migração para o subcomponente ***Manager***. O ***Manager*** é responsável por gerenciar a comunicação com a interface, as requisições de alocação e migração, os envios de requisição de dados do *cluster* através do *collector*, e a entrada de dados para o subcomponente de decisão.

O ***Collector*** é o componente responsável por coletar as informações do *cluster*, como detalhes de hardware e software dos *hosts* e máquinas virtuais. O ***DecisionComponent*** é o componente responsável pela tomada de decisão para os processos de alocação e migração, baseado em requisitos de software, hardware e integridade. O ***Cluster*** é o componente que

representa abstratamente a infraestrutura computacional que será utilizada pelo *cluster*.

O ***MechanismAttestation*** é um componente externo, responsável por realizar verificações periódicas no *cluster*, a fim de verificar os requisitos de integridade especificados no momento da alocação de máquinas virtuais, além de verificar o estado de integridade dos *hosts* para decisão de migração ou não das máquinas virtuais, com base nos requisitos estabelecidos. A base de dados será responsável por armazenar as informações das máquinas virtuais alocadas, e o estado de integridade dos *hosts* e máquinas virtuais associadas.

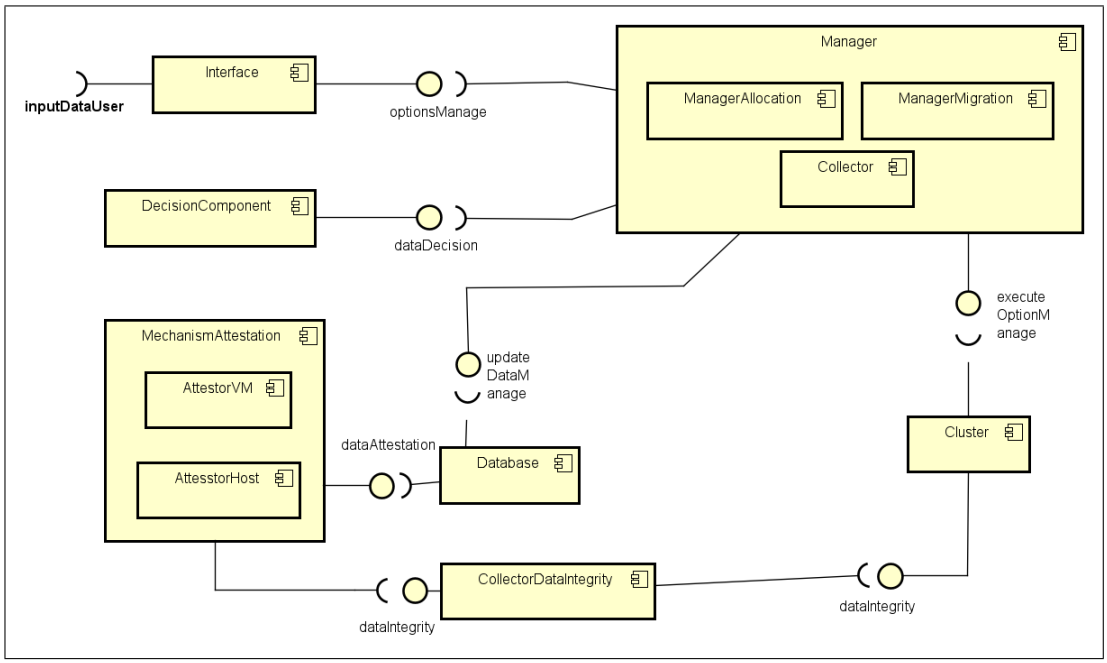


Figura 3.1: Diagrama Arquitetural da abordagem proposta.

3.2 Arquitetura

A arquitetura da ferramenta pode ser observada na Figura 3.1 que detalha os componentes envolvidos e os principais serviços fornecidos por cada componente para realização dos processos de alocação e migração. Dentro da arquitetura, a Tool Manager precisará de comunicação direta com o nó central do ***Cluster*** para solicitação de recursos de hardware e por meio da utilização de SDKs ou APIs, a depender da tecnologia utilizada no *cluster*, realizar as chamadas de criação e migração de máquinas virtuais.

Além disso, será necessário realizar consultas à base de dados que serão alimentadas

por outros dois componentes externos: o ***Collector***, responsável por realizar a varredura dos demais componentes e atualizar a base de dados e o ***MechanismAttestation***, responsável pelos dados da atestação. A ***ToolManager***, também realiza a comunicação com a base de dados, por meio da abstração realizada pelo ***ModelDAO***, que se comunica diretamente com a base de dados, abstraindo o processo de comunicação e requisições da camada acima.

3.2.1 Descrição dos componentes

3.2.1.1 Tool Manager

A *Tool Manager* é o principal componente da ferramenta proposta, incorporando os subcomponentes componente de interface, gerenciador e de decisão.

Interface

A **Interface** é o componente responsável pelo processo de interação com o usuário ou entidade externa que iniciará a cadeia de comunicação com a ferramenta. Na Figura 3.2, é possível observar o processo inicial de comunicação da interface com os demais componentes. A interface fornecerá uma lista de serviços disponíveis para gerenciamento das operações intermediadas no *cluster*.

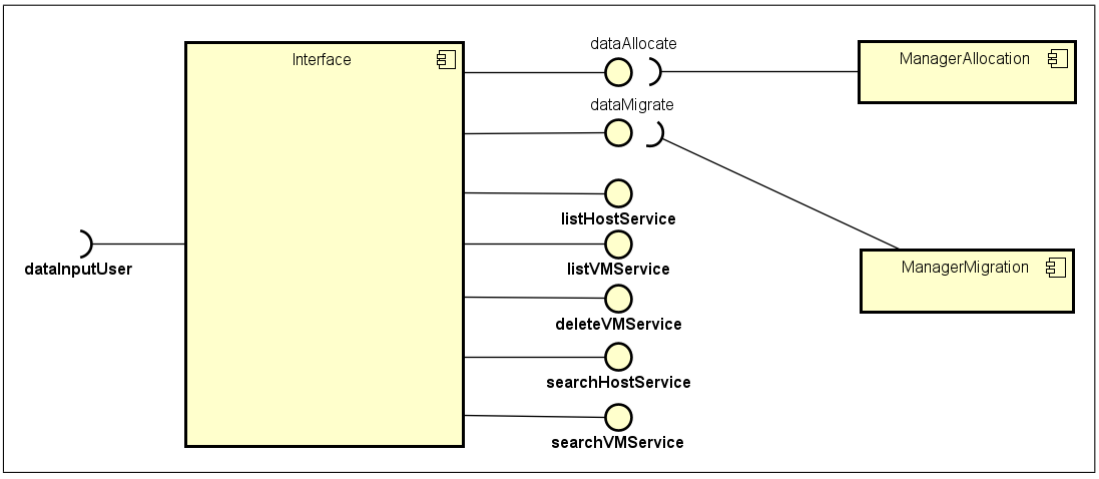


Figura 3.2: Arquitetura de Comunicação - InterfaceComponent e ManagerComponent

A interface fornecerá a seguinte lista de serviços:

- **AllocateVMService**: O serviço de alocação será responsável por receber solicitações de alocação de máquinas virtuais.

  - Parâmetros:
    1. Nome da máquina virtual;
    2. Número de núcleos da CPU necessários (valor inteiro);
    3. Memória RAM necessária (valor inteiro em GB);
    4. Opção de armazenamento: se será o *datastore* padrão do *host* ou o *storage* compartilhado (valor textual);
    5. Tipo de sistema operacional (valor textual, podendo ser *windows* ou *linux*);
    6. Versão do sistema operacional (valor textual);
    7. HostAttested (valor textual, variando entre *Y - representando true ou 1* e *N - representando false ou 0*);
    8. VMAttested (valor textual, variando entre *Y - representando true ou 1* e *N - representando false ou 0*).
  - Retorno: valor *booleano - true* ou *false*.

- **MigrateVMService**: Este é o serviço de migração manual onde o usuário ou serviço externo realizar a migração de uma máquina virtual de um *host* de origem para um *host* de destino.
  - Parâmetros
    1. Nome da máquina virtual;
    2. Nome do *host* de destino.
  - Retorno: valor *booleano - true* ou *false*.

- **MigrateAutomaticVMService**: O serviço de *MigrationAutomatic* corresponde à migração automática realizada pela abordagem proposta. Semelhante ao serviço anterior, esta migração também consiste na migração de máquina de um *host* de origem - que deixou de atender aos requisitos - para um *host* de destino - que atualmente possui requisitos que correspondem aos requisitos originalmente especificados para a máquina virtual. Essa função não recebe parâmetros e não há retorno. A migração automática é um subserviço do serviço de migração e é realizada de forma automática através da *Tool Manager*.

- **ListHostsService**: O serviço consiste em solicitar ao *collector* informações dos *hosts* presentes no *cluster*. listando todos os dados dos *hosts* retornados pelo *cluster*, especificamente informações de hardware - disco, CPU, memória, modelo de CPU, UUID.
  - Parâmetros:
    1. Não há entrada.
  - Retorno: lista de objetos.
- **ListVMService**: O serviço consiste em solicitar ao *collector* informações das máquinas criadas e presentes no *cluster*, recuperando informações de hardware como - disco, CPU, memória, modelo de CPU, UUID - e de software, como nome tipo e versão do sistema operacional, nome da VM, UUID, path, tipo de *host*.
  - Parâmetros:
    1. Não há entrada.
  - Retorno: lista de objetos.
- **DeleteVMService**: O serviço consiste em remover máquinas virtuais do *cluster* por opção do usuário ou sistema externo.
  - Parâmetros:
    1. Nome da máquina virtual.
  - Retorno: valor *booleano* - *true* ou *false*.
- **SearchVMService**: O serviço consiste em buscar informações de um dado ou máquina virtual no *cluster*, retornando detalhes de hardware e software.
  - Parâmetros:
    1. Nome da máquina virtual, id gerado no *cluster* ou UUID da máquina.
  - Retorno: valor *booleano* - *true* ou *false*.
- **SearchHostService**: O serviço consiste em buscar informações de um dado *host* no *cluster*, retornando detalhes de hardware e software.
  - Parâmetros:

1. Nome da *host*, id gerado no *cluster* ou UUID da máquina.

– Retorno: valor *booleano* - *true* ou *false*.

DecisionComponent

O *DecisionComponent* é o componente responsável pela tomada de decisão com base em requisitos de hardware, software e de confiabilidade. O processo de decisão é melhor detalhado na seção **??**. De forma breve, o processo de decisão será iniciado a partir de uma solicitação do *ManagerAllocation* ou do *ManagerMigration*, que solicitará ao componente de decisão um **melhor *host*** e um **melhor *storage***.

O componente de decisão realizará a comunicação com o subcomponente do *Manager*, o *Collector* que irá se comunicar com o *cluster* e recuperar os recursos de hardware e software. Simultaneamente, o componente de decisão irá requisitar ao componente externo *MechanismAttestation* os dados de confiabilidade dos *hosts* e das máquinas virtuais. Com todos os dados disponíveis, é possível realizar a decisão considerando todos os parâmetros informados no momento da alocação ou através da verificação da ferramenta para a realização de uma migração automática.

Manager

O *Manager* é subdivido em três subcomponentes:

ManagerAllocation

Este componente é responsável por gerenciar todo o processo de alocação de VMs. Quando inicializado, através da interface da ferramenta, é solicitado ao gerenciador que realize a alocação de uma máquina virtual. Como mostrado na Figura 3.3, quando o *ManagerAllocation* recebe como entrada um valor *booleano* para realizar a alocação, será enviada uma mensagem de volta à interface para requisitar dados de entrada, como nome da VM, parâmetros de hardware e software e requisitos de confiabilidade.

Após receber os dados, o gerenciador de alocação envia uma requisição ao *DecisionComponent* para realizar a escolha do melhor *host* e melhor *storage* - quando for o caso - com base nos requisitos solicitados. Quando da escolha do host e storage, o gerenciador de alocação constrói a*request* ao *cluster* para realizar a alocação da máquina virtual, e atualizada a base de dados com os requisitos de confiabilidade de máquina virtual e *host*.

ManagerMigration

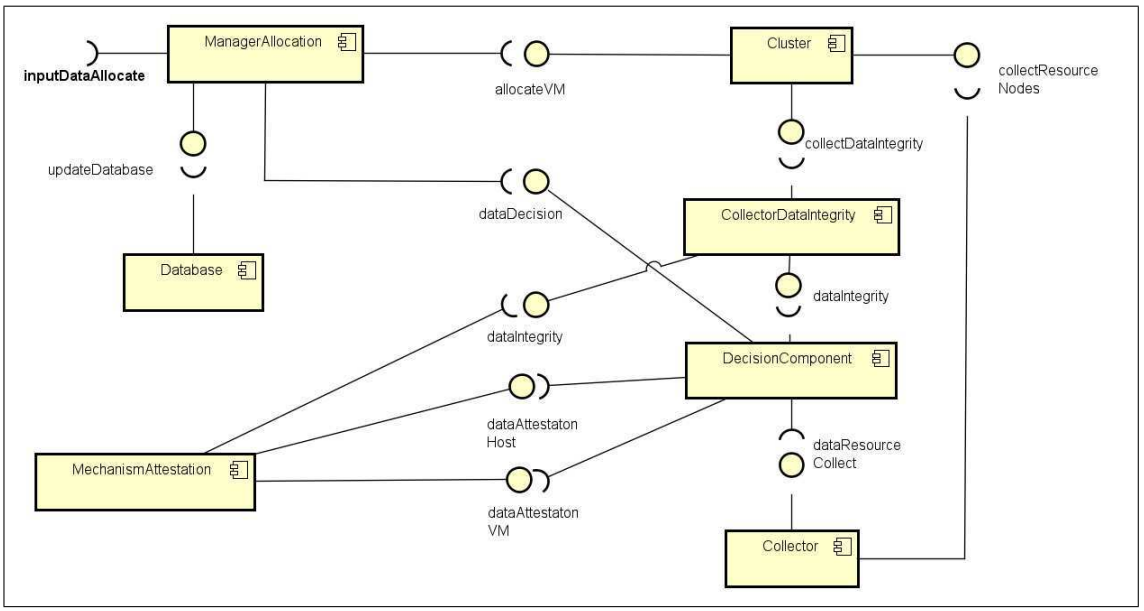


Figura 3.3: Arquitetura de Comunicação - ManagerAllocation

O *ManagerMigrate* é responsável por realizar verificações contínuas no *cluster* e compará-las com os valores de confiabilidade inicialmente estabelecidos e armazenados na base de dados. Este componente também pode funcionar de forma manual, através de um início direto pela interface, que forcará a verificação do *cluster* no momento em que foi solicitado. Quando esse componente é iniciado, é enviada uma requisição ao *DecisionComponent*, conforme demonstrada na figura 3.4, que recupera os dados de atestação de *host* e máquina virtual fornecidos pelo *MechanismDecision*, o resultado dessa interação são os dados de confiabilidade. Após obter esses dados, é requisitado ao *Collector* a recuperação dos recursos específicos de hardware, software, rede e *storage*. Após a decisão, o item escolhido é retornado e realizado o processo de migração, e o mesmo processo ocorre para a quantidade de VM que houver no *cluster* para serem migradas, caso o seu requisito confiabilidade ou recursos tenham sido violados.

Collector

O *collector* é um subcomponente responsável por realizar a coleta dos dados de hardware e software do *cluster*. Quando o *ManagerAllocation* ou o *ManagerMigrate* recebe uma requisição de um sistema externo ou usuário, o *DecisionComponent* recebe uma requisição e envia uma *request* para o *Collector* e para o *CollectorDataIntegrity* para recuperar os recursos de hardware, software e confiabilidade dos *hosts* e máquinas virtuais. Assim, o *collector*

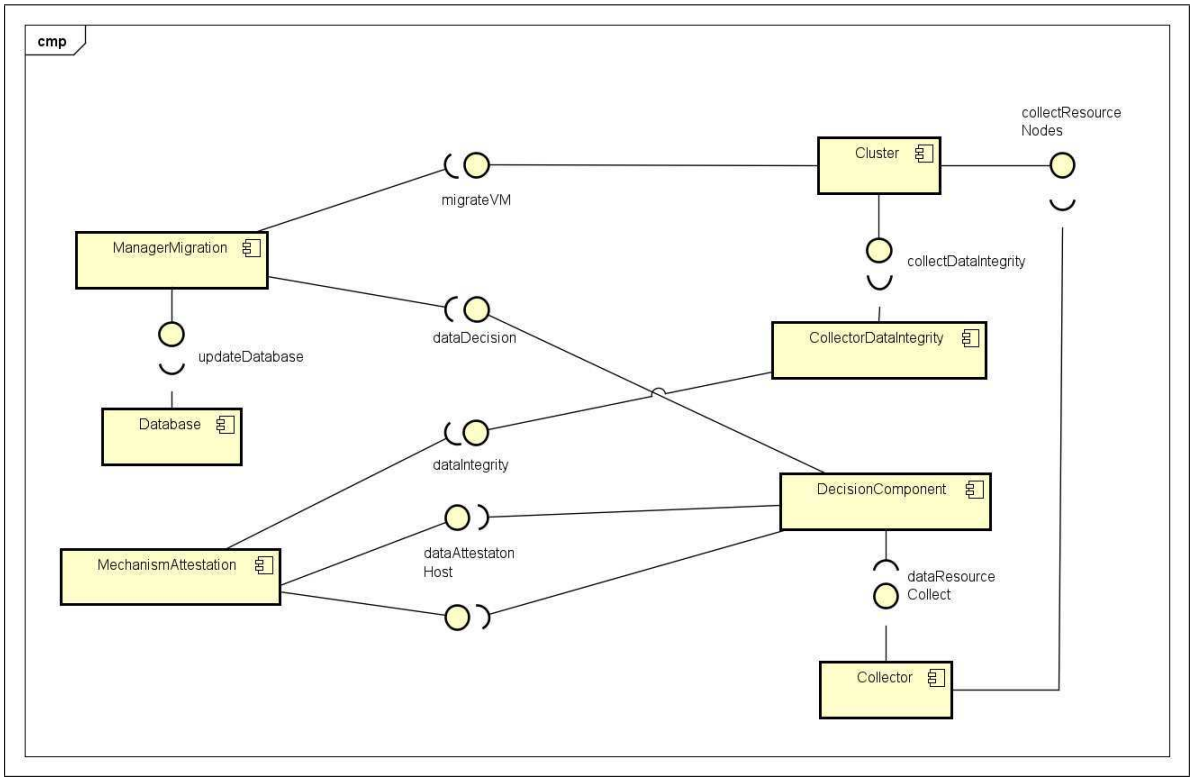


Figura 3.4: Arquitetura de Comunicação - ManagerMigration

funciona recebendo requisitos e retornando os resultados da coleta para o componente de decisão.

3.2.1.2 Cluster

O *cluster* é uma abstração para um componente externo responsável pelo gerenciamento de um conglomerado de hospedeiros (*hosts*) e máquinas virtuais. Este componente é responsável por fornecer todos os dados de hardware, software, rede e armazenamento necessário à utilização dos demais componentes. A comunicação com o *cluster* acontecerá por meio da utilização de *APIs* ou *SDKs* públicos, e quando da impossibilidade de utilização, a comunicação acontecerá por *command line* através da interação direta com o nós do *cluster*. Nesta implementação, o *cluster* fornecerá opções de recuperação de dados diversos dos nós, criação e migração de máquinas virtuais, assim como listagem e busca de VMs e *hosts*.

3.2.1.3 Collector Data Integrity

O componente *collectorDataIntegrity* é um componente externo, responsável por realizar a coleta dos dados de integridade dos *hosts* e máquinas virtuais do *cluster*. Para este trabalho, são considerados dados de integridade os dados do TPM, especificamente a lista atualizada dos valores de *Platform Configuration Register (PCR)* e o log de eventos. Para os *hosts*, o log de eventos dependerá do virtualizador utilizado, e contará com uma implementação específica para cada fabricante, quando estes dados não puderem ser obtidos de forma automática pelo nó gerenciador do *cluster*.

Para as máquinas virtuais são coletados arquivos específicos para cada categoria de sistema operacional, para sistemas baseados em Linux são coletados o *Integrity Measurement Architecture (IMA)*, para sistemas Windows serão coletados o *Windows Boot Configuration Log (WBCL)*. O *collector* se comunicará com os nós do *cluster* através de *APIs* e *SDKs* ou requisição direta de comandos usando *command line*, quando não houver *API* pública para comunicação.

3.2.1.4 MechanismAttestation

AttestorVM

Este componente é responsável por coletar os dados de integridade de máquinas virtuais com TPM e realizar o processo de atestação. O processo de coleta leva em considerados os valores atualizados de *PCRs* do TPM da máquina, e o arquivo de log de eventos correspondente ao sistema operacional, se for uma VM Windows será coletado o *WBCL*, se for uma VM Linux será coletado o *IMA*.

A partir da coleta desses dados, é iniciado o processo de atestação que consiste em verificar o *boot aggregate* e o PCR dinâmico, que também variam conforme o sistema operacional, mas de modo geral o *boot aggregate* consiste em um *hash* em *SHA1* ou *SHA256* de um conglomerado de *PCRs*, para sistemas Linux por exemplo, utiliza-se os valores de 0-7.

A verificação realizada serve para identificar se houve alguma violação dos *PCRs* estáticos durante o intervalo entre *boots* da máquina. Como os *PCRs* utilizados no processo de *boot aggregate* devem ser estáticos, por são gravados de fábrica pelo produtor do hardware ou pelo sistema operacional originalmente instalado, qualquer modificação nesse conjunto

de *PCRs* poderá ser mais facilmente identificada.

A segunda etapa consiste em validar o PCR dinâmico - que também varia conforme o SO -, para isso, é utilizado um *PCR* específico e o arquivo de log de eventos. Inicialmente, cada evento do arquivo possui um *hash* que corresponde ao valor atual do *PCR* mais o último valor de *hash* do evento, com esses dois valores é gerado um novo *hash*, e esse mecanismo prossegue duranta cada nova entrada no log de evento, assim é possível validar o log de eventos com os valores atuais de *PCRs*.

AttestorHost

Este componente tem um comportamento semelhante ao *AttestorVM*, porém os arquivos de entrada no processo segue um fluxo diferente. O processo de atestação de *host* ocorre em apenas uma etapa, a partir da verificação do log de eventos. O log de eventos do *host* dependerá do SO instalado e o do hipervisor conectado, em alguns casos é possível conseguir um arquivo de log de eventos com medições, mas em alguns outros casos é necessário criar um log de eventos a partir de determinadas entradas de outros arquivos de registro da máquina. Após a obtenção do log de eventos, basta verificar se o valor em *hash* do último evento correspondente ao valor mais atual do PCR dinâmico. Com a obtenção do estado de atestação do *host*, basta atualizar a base de dados com as últimas modificações.

3.2.1.5 Database

O componente *database* representa abstratamente qualquer base de dados que poderá ser utilizada para armazenar os dados da abordagem proposta. Para esta implementação, uma base de dados adequada deverá ser capaz de criar, buscar, atualiza e remover dados.

3.3 Modelo de Dados

A base de dados de atestação contém dados relacionados ao último estado de atestação de *hosts* e máquinas virtuais, bem como dos requisitos de confiabilidade estabelecidos pelo usuário no momento da alocação de máquinas virtuais. Essa base é consumida durante o processo de alocação e migração para tomada de decisão considerando o nível de requisitos de segurança.

A base de dados é alimentada por vários componentes, o *MechanismAttestation* é res-

ponsável por realizar a atualização dos dados relacionados à confiabilidade, resultantes do processo de atestação de VMs e *hosts*. O *ManagerAllocation* realiza inserções na base de dados, com os requisitos de confiabilidade inicialmente estabelecidos para máquinas virtuais e *host* relacionado, no momento da alocação. O *ManagerMigration* realiza atualização na tabela de *Migration_Requirements* e na tabela *VirtualMachineAttestation*, a partir dos dados resultantes das decisões realizadas para a execução da migração. Nesse processo, máquinas virtuais tiveram seu estado de confiabilidade inicial alterado, dessa forma, quando realiza-se a migração, é necessário atualizar as tabelas novamente com os novos requisitos e com a atualização do novo *host*.

Também será possível persistir informações específicas do tipo de atestação que está sendo realizada: ***1 - Completa:*** com a atestação do chip TPM e a atestação do arquivo de medições; ***2 - Parcial:*** apenas a atestação do chip TPM. O arquivo de medição é dependente do sistema operacional que será utilizado, sistemas baseados em Linux utilizam a *Integrity Measurement Architecture (IMA)* e sistemas Windows utilizam o arquivo gerado no próprio sistema chamado de *Windows Boot Configuration Log (WBCL).*

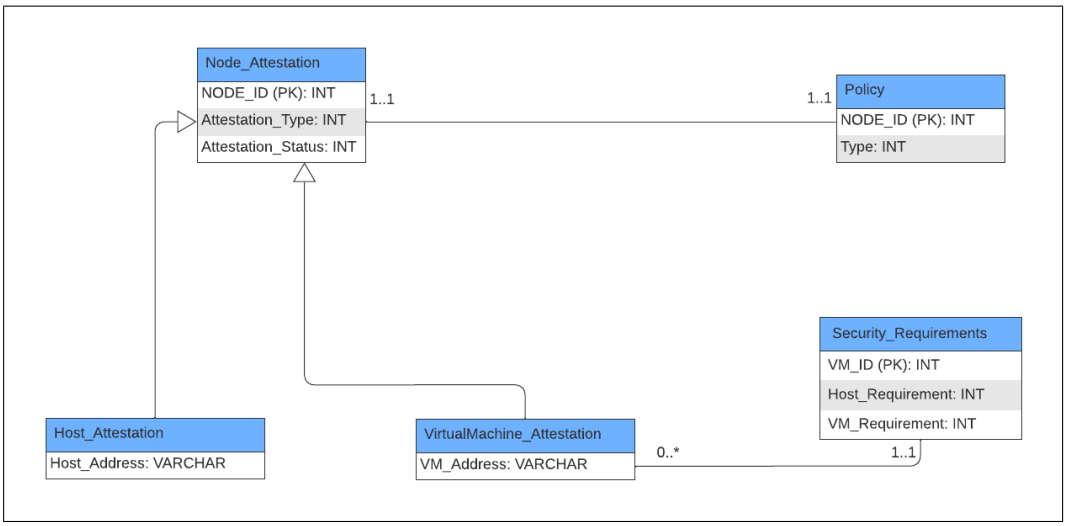


Figura 3.5: Diagrama Entidade-Relacionamento da Database Attestation

A base de dados é composta por cinco tabelas, conforme mostrado na Figura 3.5. A tabela *Host_Attestation* guarda dados relacionados aos *hosts* do *cluster*, assim como a tabela *VirtualMachine_Attestation* guarda informações de todas as máquinas virtuais monitoradas pelo mecanismo de atestação. As duas tabelas herdam da tabela *Node_Attestation* que possui

atributos compartilhados pelos *hosts* e as máquinas virtuais. A tabela *Security_Requirements* guarda os dados relacionados aos requisitos de segurança estabelecidos pelos usuários no momento de alocação de máquinas virtuais. Por fim, a tabela *Policy* guarda dados relacionados à política de manutenção da máquina virtual em caso de o ambiente deixar de ser confiável.

Para a **máquina virtual** são possíveis quatro valores de ações:

1. pause;
2. delete (destroy);
3. noop; e
4. continue.

Para o ***host***, a tabela é usada para decidir sobre a disponibilidade ou não de alocação de novas VMs. Podendo ter três ações possíveis:

1. pause;
2. noop; e
3. continue.

3.4 Processo de Coleta de Dados

3.4.1 Coleta de Recursos do Cluster

O processo de coleta de dados compreende dados sobre os componentes do *cluster*. As tabelas de *Host_Attestation* e *VirtualMachine_Attestation* são inicialmente preenchidas com o *id* do nó, seja *host* ou máquina virtual. O *collector* é inicializado a cada chamada ao processo de alocação ou de migração, para atualização da base de dados com novas entradas de nós.

A Figura 3.6 exibe o processo de coleta de dados realizada pelo componente *Collector*. Atualmente, o componente de coleta é inicializado a cada iteração de alocação ou migração, mas pode-se definir um *time* para que o processo realize a coleta e atualize a base de dados.

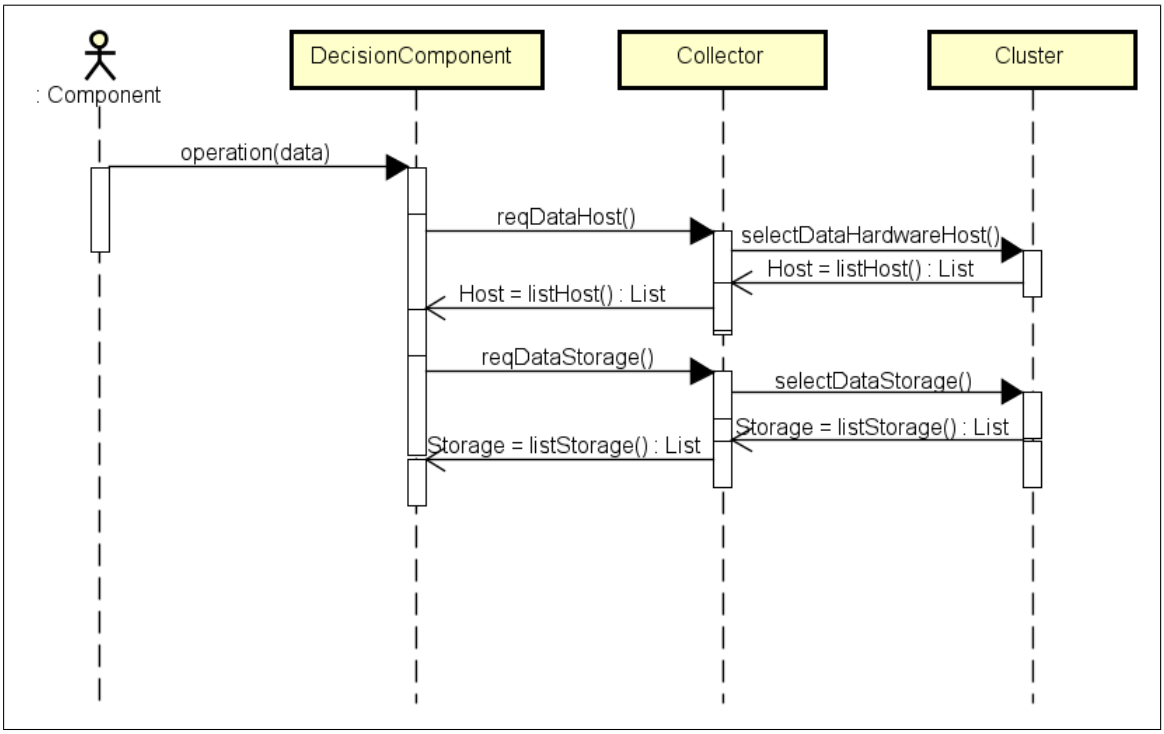


Figura 3.6: Processo de coleta de recursos dos nós.

De modo geral, sempre que há uma requisição à interface da *Tool Manager*, o *Manager* realiza uma requisição ao *Collector* para atualizar a base de dados. O *Collector* envia uma requisição de volta para o *Manager* solicitando os dados do cluster: *hosts*, máquinas virtuais e os *ids* de cada nó, esses dados são enviados para o *Collector* que realiza uma comparação com os dados já inseridos na base, se algum nó não está presente, então esse nó é adicionado com as informações passadas pelo *Manager*.

3.4.2 Coleta de Dados do Confiabilidade

A coleta de dados de confiabilidade consiste na coleta de dados de atestação fornecidos pelo componente *MechanismAttestation*, e solicitada pelo *DecisionComponent*. Inicialmente, quando é solicitada alguma operação ao *Manager*, seja de alocação ou migração, é realizado uma *request* ao componente de decisão para retorno do melhor *host* e *storage*.

O componente de decisão então realiza uma requisição ao *Collector* e ao *MechanismAttestation* para recuperar os dados de confiabilidade dos *hosts* e máquina virtual - quando for o caso de migração, como mostra a Figura 3.7. No que tange aos dados de confiabilidade, o

mecanismo de atestação realiza uma chamada ao *CollectorDataIntegrity* para recuperar do *cluster* os dados do TPM e do log de eventos dos *host* e das máquinas virtuais, para realizar o processo de atestação de integridade que retornar os status de confiabilidade para os nós envolvidos.

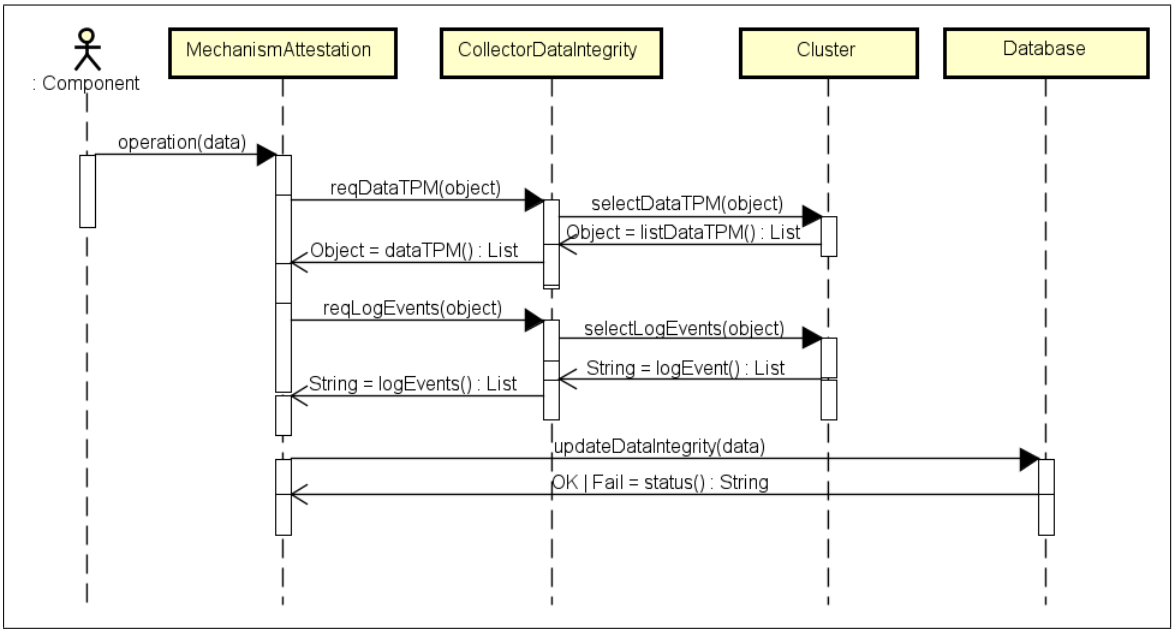


Figura 3.7: Processo de coleta de dados de confiabilidade.

Quando os dados de confiabilidade são recebidos, são retornados para o componente de decisão, e o *MechanismAttestation* realiza uma inserção ou atualização da base de dados, a depender da operação original que será realizada.

3.5 Tomada de decisão para o processo de alocação

3.5.1 Descrição

O processo de alocação consiste na criação de máquinas virtuais dentro de um *cluster*, com base em três níveis de requisitos: 1) hardware, 2) software e 3) confiabilidade. O nível de *hardware* consiste em requisitos específicos do hardware que devem ser setados para a criação do hardware emulado da máquina virtual. O nível de *software* está relacionado ao tipo de sistemas operacionais e a versão/distribuição que será utilizada. O nível de confiabilidade

está relacionado aos requisitos de integridade, aferidos através de um processo de atestação, da máquina virtual e do *host* em que a máquina será alocada.

3.5.2 Níveis de Requisitos

A criação de máquinas virtuais dentro da ferramenta é realizada através do processo de alocação, que leva em consideração três níveis de requisitos:

1) **Requisitos de hardware**: são os requisitos relacionados aos recursos de hardware para criação da máquina virtual e compreendem:

- Nome da máquina virtual: valor texto.
- Número de CPUs: valor inteiro.
- Tamanho de memória em GB: valor inteiro.
- *Storage* compartilhado ou disco local do *host*: *storage* ou local.
- Tamanho do disco em GB: valor inteiro.
- *Datacenter*: valor texto - setado por padrão.

2) **Requisitos de software**: compreende o tipo de sistema operacional, o nome e a versão ou distribuição.

- Tipo de sistema operacional: valor texto.
- Distribuição: valor texto (se aplicável)
- Versão: valor texto.
  - Setado com base na tabela de *guest id* do *cluster*.

3) **Requisitos de confiabilidade**

- Requisito de atestação de máquina virtual: valor inteiro (0 ou 1).
- Requisito de atestação de *host*: valor inteiro (0 ou 1).

3.5.3 Tomada de Decisão

A tomada de decisão no processo de alocação consiste em avaliar os recursos disponíveis dentro do *cluster* e os requisitos de confiabilidade com base nos dados de atestação. O processo de alocação inicia quando um usuário realiza uma requisição de alocação de uma máquina virtual dentro do *cluster*. Os requisitos base de hardware, software e confiabilidade são solicitados ao usuário.

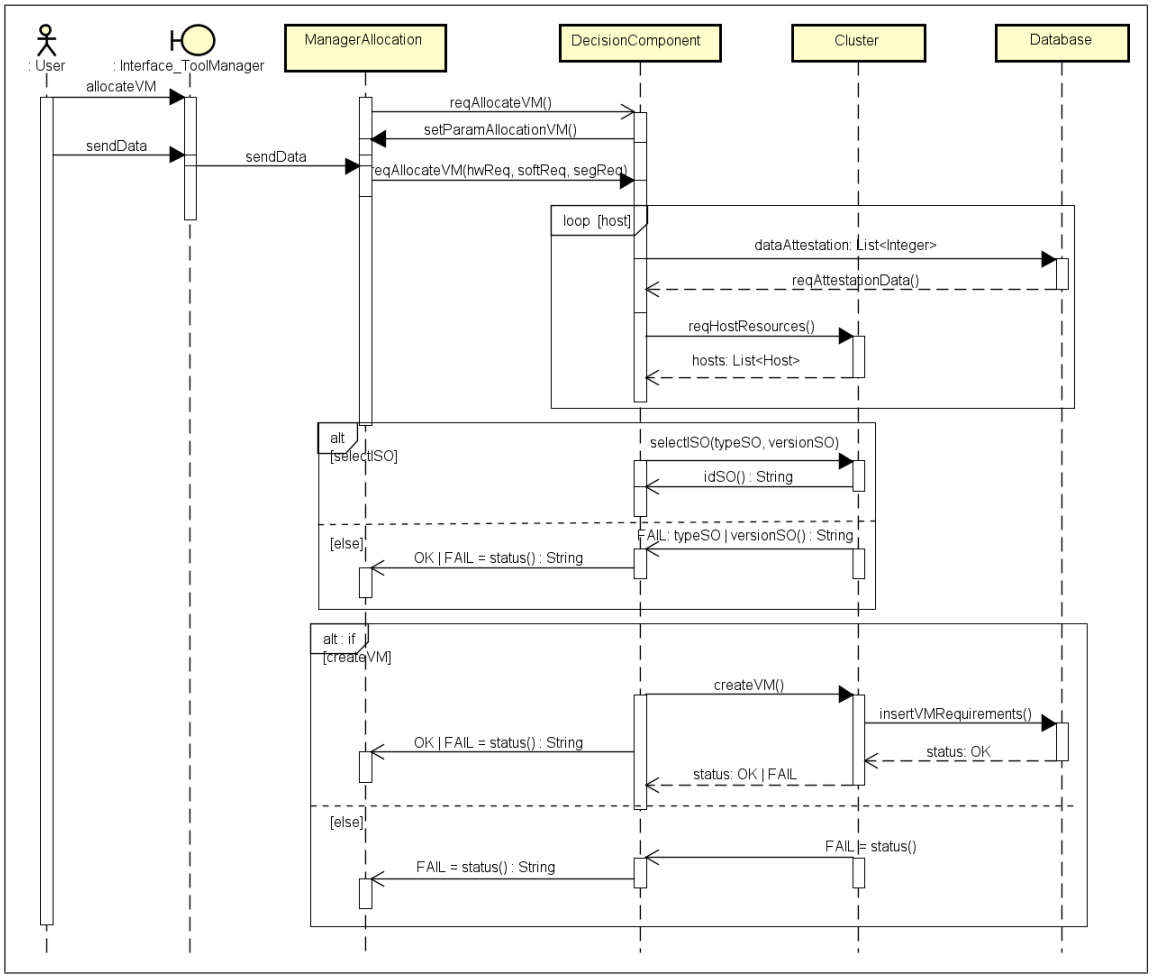


Figura 3.8: Diagrama de Sequência do Processo de Alocação.

Com base nos requisitos fornecidos, a ferramenta inicia a escolha do melhor *host* e do melhor *storage* - a depender da escolha do usuário - que ocorre em algumas etapas, conforme pode ser observado na Figura 3.8 e descrito das etapas abaixo. O processo inicia realizando a verificação do requisitos de confiabilidade do *host*.

Para o requisito de confiabilidade do *host* = *true*, temos:

1. **Requisito de segurança**

   (a) Existe apenas um *host* no *cluster*: selecionado → etapa 2).

   (b) Existe uma lista de *host* disponíveis no *cluster*: uma sub lista é selecionada → etapa 2).

   (c) Não existe um *host* disponível: a VM não será alocada → processo é finalizado.

2. **Requisito de hardware**

   (a) *Hosts* que atendem simultaneamente os requisitos de número de CPUs e memória

      i. Existe apenas um *host* no *cluster*: selecionado → etapa 3).

      ii. Existe uma lista de *host* com recursos disponíveis: o último a ser iterado é escolhido → etapa 3).

      iii. Não existe um *host* disponível: a VM não será alocada → finalizado.

   (b) *Host* que só atendem o requisito de memória

      i. Existe apenas um *host*: selecionado → etapa 3).

      ii. Existe uma lista de *host*: o último a ser iterado é escolhido → etapa 3).

      iii. Não existe um *host* disponível: a VM não será alocada → finalizado.

3. **Requisito de software**

   (a) O *guest id* fornecido é válido: selecionado → VM criada.

   (b) O *guest id* id fornecido é incompatível com o tipo de sistema operacional → erro → processo finalizado.

   (c) O *guest id* fornecido é inválido: erro → processo finalizado.

Para o requisito de integridade do *host* = *false*, temos:

1. **Requisito de integridade**

   (a) Existe apenas um *host*: selecionado → etapa 2).

   (b) Existe uma lista de *host*: uma sub lista de *hosts* é selecionada → etapa 2).

   (c) Não existe um *host* disponível com integridade = false

i. Existem *host* disponíveis com segurança = *true*: selecionado → etapa 2).

ii. Não existem *hosts* disponíveis: erro → finalizado.

2. **Requisito de hardware**

   (a) *Host* que atendem simultaneamente os requisitos de número de CPUs e memória:

      i. Existe apenas um *host*: selecionado → etapa 3).

      ii. Existe uma lista de *host*: o último a ser iterado é escolhido → etapa 3).

      iii. Não existe um *host* disponível: a VM não será alocada → finalizado.

   (b) *Host* que só atendem o requisito de memória

      i. Existe apenas um *host*: selecionado → etapa 3).

      ii. Existe uma lista de *host*: o último a ser iterado é escolhido → etapa 3).

      iii. Não existe um *host* disponível: a VM não será alocada → finalizado.

3. **Requisito de software**

   (a) O *guest id* fornecido é válido: selecionado → VM criada.

   (b) O *guest id* fornecido é incompatível com o tipo de sistema operacional → erro → processo finalizado.

   (c) O *guest id* fornecido é inválido: erro → processo finalizado.

3.6 Tomada de decisão para o processo de migração

3.6.1 Descrição

O processo de migração automática pode ser iniciado pelo usuário ou por verificação automática da ferramenta. A verificação automática tem o objetivo de analisar os requisitos de confiabilidade definidos pelo usuário no momento em que as máquinas virtuais foram alocadas e verificar se estão coerentes com o estado atual das máquinas virtuais ou *host*, caso o status de confiabilidade esteja diferentes, indicará que as máquinas virtuais ou o *host* tiveram sua confiabilidade violada.

A migração também poderá ser executada no caso da verificação automática indicar que os recursos do *host* estejam no limite de utilização, o que poderá ocasionar o mal funcionamento da máquina virtual, nesse caso, a máquina virtual também poderá ser migrada.

Durante a análise dos requisitos, a migração automática poderá ser realizadas em três situações:

1. A VM foi criada com o requisito de segurança de *host = 1*, caso o *host* que ela foi inicialmente alocada tenha esse status de alterado, então ocorrerá migração automática dessa VM para outro *host* atestado.

2. A VM foi criada com o requisito de segurança de *host = 0*, caso o *host* que ela foi inicialmente alocada tenha esse status alterado, então ocorrerá a migração automática dessa VM para outro *host* não atestado.

3. A VM está alocada em um *host* com indicativo de escassez de recursos de hardware.

As etapas da tomada de decisão estão enumeradas na subseção seguinte, e o diagrama da Figura 3.9 mostra a sequência de eventos disparadas entre os objetos participantes do processo de migração automática.

3.6.2 Etapas da Tomada de Decisão

O processo de migração leva em consideração a tabela *Security_Requirements*, a tabela *Host_Attestation* e a tabela *VirtualMachine_Attestation* e os recursos dos nós do *cluster*. A partir da Figura 3.9, podemos visualizar a seguinte sequência de atividades:

1. Seleciona todas as VMs em que o estado de confiabilidade do *host* atual não está compatível com o estado de atestação desejado para aquela VM no momento de alocação, e que está armazenado na tabela *Security_Requirements*. A partir das VMs selecionadas, as próximas etapas ocorrem em um *loop* para cada VM.

2. Se houver VMs com requisito de atestação de *host = 1* para serem migradas:

   (a) Seleciona todos os demais *hosts* com status de atestação = 1.

   (b) Inicia o processo de escolha do melhor *host*.

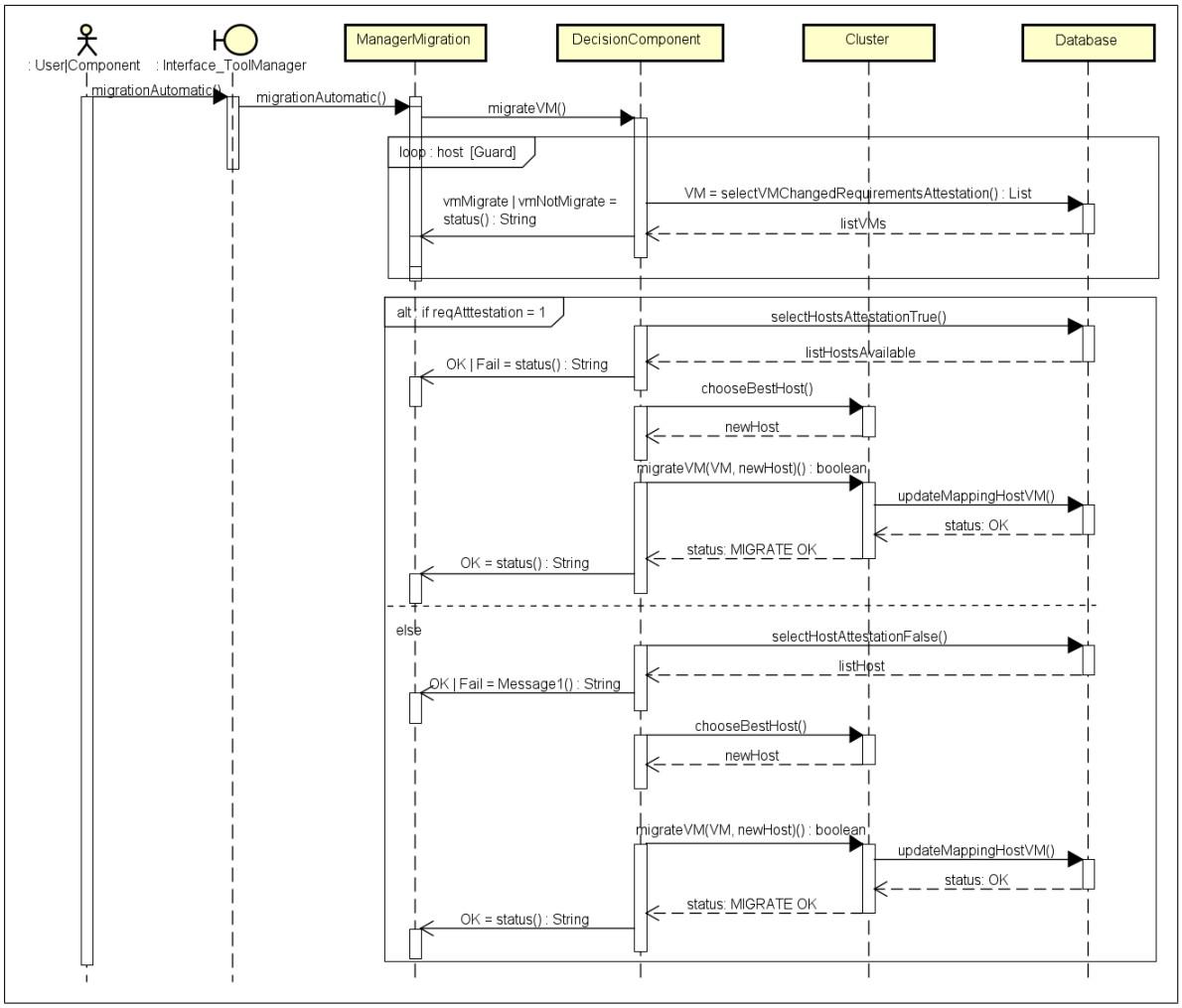


Figura 3.9: Diagrama de Sequência do Processo de Migração.

i. Seleciona todos os *hosts* que sejam do mesmo *server model* do *host* alocado originalmente para a VM que será migrada - para evitar incompatibilidade hardware quando for realizada a migração.

ii. Seleciona todos os **hosts** entre os escolhidos na etapa anterior que satisfaça simultaneamente os requisitos de recursos *hardware* de memória e número de CPUs.

iii. Caso o passo anterior não seja realizado com sucesso, então não é possível escolher um *host* que atenda simultaneamente os dois requisitos, então é escolhido um *host* que atenda ao requisito de hardware de maior memória.

iv. Caso nenhum *host* seja selecionado, então não há no *cluster* outro *host* disponível que satisfaça o nível de hardware. Nesse caso, o processo de migração é finalizado com erro por falta de recursos compatíveis com os requisitos originais da máquina virtual.

(c) Realiza a migração automática para o novo *host* ou mensagem de erro.

3. Se houver VMs com requisitos de atestação de *host = 0* para serem migradas:

(a) Seleciona todos os demais *hosts* com status de atestação = 0.

(b) Inicia o processo de escolha do melhor *host*.

i. Seleciona todos os *hosts* que sejam do mesmo *server model* do *host* alocado originalmente para a VM que será migrada - para evitar incompatibilidade hardware quando for realizada a migração.

ii. Seleciona todos os **hosts** entre os escolhidos na etapa anterior que satisfaça simultaneamente os requisitos de recursos *hardware* de memória e número de CPUs.

iii. Caso o passo anterior não seja realizado com sucesso, então não é possível escolher um *host* que atenda simultaneamente os dois requisitos, então é escolhido um *host* que atenda ao requisito de hardware de maior MEMÓRIA.

iv. Caso nenhum *host* seja selecionado, então não há no *cluster* outro *host* disponível que satisfaça o nível de hardware. Nesse caso, o processo de migração é finalizado com erro por falta de recursos compatíveis com os requisitos

originais da máquina virtual.

(c) Realiza a migração automática para o novo *host* ou mensagem de erro.

3.7 Modelo de ameaças

3.7.1 Modelo de ameaças para computação confiável

O modelo de ameaças para a computação confiável assume que o atacante poderá ter controle total a todos os recursos do ambiente. Com a possibilidade de privilégios de superusuário poderá ter acesso a toda a pilha de serviços e infraestrutura a qual o nó atacado faz parte, incluindo acesso aos softwares, sistemas operacionais, hipervisores e a plataforma de computação em nuvem. A partir do acesso, o invasor pode alterar códigos ou arquivos binários com objetivos diversos desde o furto dos dados da infraestrutura, monitoramento do tráfego e acesso aos dispositivos.

Segundo o *Confidential Computing Consortium (CCC)*, em um artigo publicado em 2022, dentro da computação confiável é possível definir vetores de ameaça que podem explorar vulnerabilidades específicas dentro do ambiente [9]. Alguns dos principais podem ser destacados abaixo:

- **Ataques de software**: incluem ataques a softwares e *firmwares* instalados no *host*, incluindo sistema operacional, hipervisor, BIOS e outros softwares e cargas de trabalho associadas.

- **Ataques de protocolo**: inclui ataques a protocolos associados ao processo de atestação, incluindo a carga de trabalho e o transporte de dados. Nesse tipo de ataque, mesmo que o protocolo não seja comprometido, pode haver uma vulnerabilidade no provisionamento na carga de trabalho ou dados associados.

- **Ataques criptográficos**: esse tipo de ataque explora vulnerabilidades relacionadas ao uso de algorítimos primitivos ou exploração de novas técnicas de quebra criptográfica que envolva alto poder computacional.

- **Ataques físicos básicos**: esses tipos de ataques incluem extração *DRAM fria*, barramento, cache monitoramento e conexão de dispositivos de ataque em uma porta

existente, por exemplo, *PCIe*, *Firewire*, *USB-C*.

- **Ataques básicos à cadeia de suprimentos *upstream***: são ataques que consideram mudanças mais grosseiras dentro do escopo, como por exemplo adicionar portas de depuração.

3.7.2 Requisitos de Segurança

A modelagem de ameaça considerou três cenários principais levando em consideração o processo de alocação e o processo de migração. Para o cenário de alocação os requisitos considerados levaram em consideração os

Cenário 1 - Processo de Alocação

Requisitos de Segurança (Mitigações)

- Imagem de criação segura (*hash* do *.iso* tem que ser igual ao de origem);
- Rede conectada de forma segura (autenticação e criptografia);
- *Host* íntegro com status de atestação sem violação.

Monitoramento

- Utilização de um vTPM na VM;
- Utilização do TPM no *host*;
- Utilização de um mecanismo de atestação;
- Monitoramento contínuo de requisitos de confiabilidade.

Cenário 2 - Processo de Migração - Integridade do *Host*

Requisitos de Segurança (Mitigações)

- Utilização do chip TPM
- *Host* íntegro (valores do *hash* dos PCRs da ferramenta de atestação batendo com os valores base dos PCRs da máquina).

Monitoramento

- Realização da atestação dos *hosts* a cada 1 minuto ou quando ocorrer qualquer interação com o *host*, seja através de alocação, migração ou remoção de máquinas virtuais;
- Armazenamento na base de dados de todas as alterações no estado do *host*.

Cenário 3 - Processo de Migração - Integridade da VM

Requisitos de Segurança (Mitigações)

- Utilização do vTPM;
- VM íntegra (valores de *hash* dos PCRs da ferramenta de atestação batendo com os valores base dos PCRs da máquina);
- *Host* íntegro (valores do *hash* dos PCRs da ferramenta de atestação batendo com os valores base dos PCRs da máquina).

Monitoramento

- Realização da atestação das máquinas virtuais a cada 1 minuto ou quando ocorrer qualquer interação com o *host*, através da requisição de alguma operação como: alocação, migração ou remoção;
- Realização da atestação dos *hosts* nos quais as máquinas virtuais estão hospedadas, realizando um monitoramento a cada 1 minuto ou quando ocorrer qualquer interação com o *host*, através da requisição de alguma operação como: alocação, migração ou remoção;
- Armazenamento na base de dados de todas as alterações no estado do máquina virtual;
- Armazenamento na base de dados de todas as alterações no estado de integridade do *host*.

3.7.3 Cenários de ameaças

O modelo de ameaça foi considerado sob a ótica de três cenários: (1) cenário da alocação, observado na Figura 3.10; (2) cenário de migração considerado a integridade do *host*, observado na Figura 3.11; e (3) cenário de migração considerando a integridade da máquina virtual, observado na Figura 3.12.

O cenário (1) considera as possíveis vulnerabilidades encontradas diante do processo de alocação de uma máquina virtual. Nesse cenário, o alvo seria a máquina virtual em processo de alocação onde o atacante seria um hacker com objetivo de comprometer a máquina virtual, podendo utilizar como meio a rede em que a comunicação da máquina virtual está acontecendo ou o arquivo de imagem *.iso* para a criação da máquina virtual. No processo, onde pode ocorrer o comprometimento do arquivo original através de injeção ou modificação para introduzir ou explorar uma vulnerabilidade previamente identificada no arquivo de imagem.

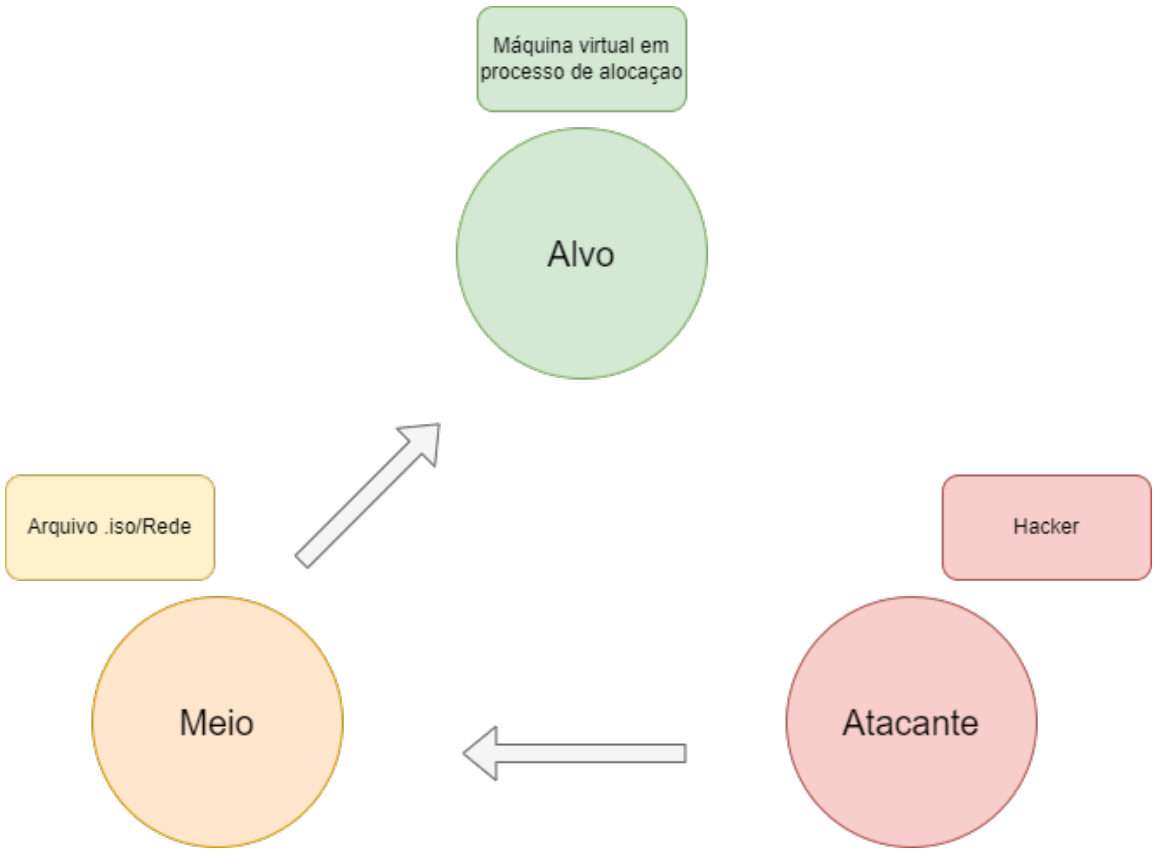


Figura 3.10: Cenário 1 - Processo de Alocação

O cenário (2) considera as possíveis vulnerabilidades encontradas diante do processo de migração de uma máquina virtual partindo da perspectiva da integridade do *target host* ao qual a VM está sendo migrada. Nesse cenário, o alvo seria o *target host* em execução e o atacante seria um hacker com objetivo de comprometer o novo hospedeiro da máquina virtual em execução. Para isso, poderá utilizar como meio a *API* de comunicação do ambiente de *cloud*, acesso via *SSH* ou acesso ao *datastore* compartilhado em são armazenados dados dos *hosts* e das máquinas virtuais hospedadas.

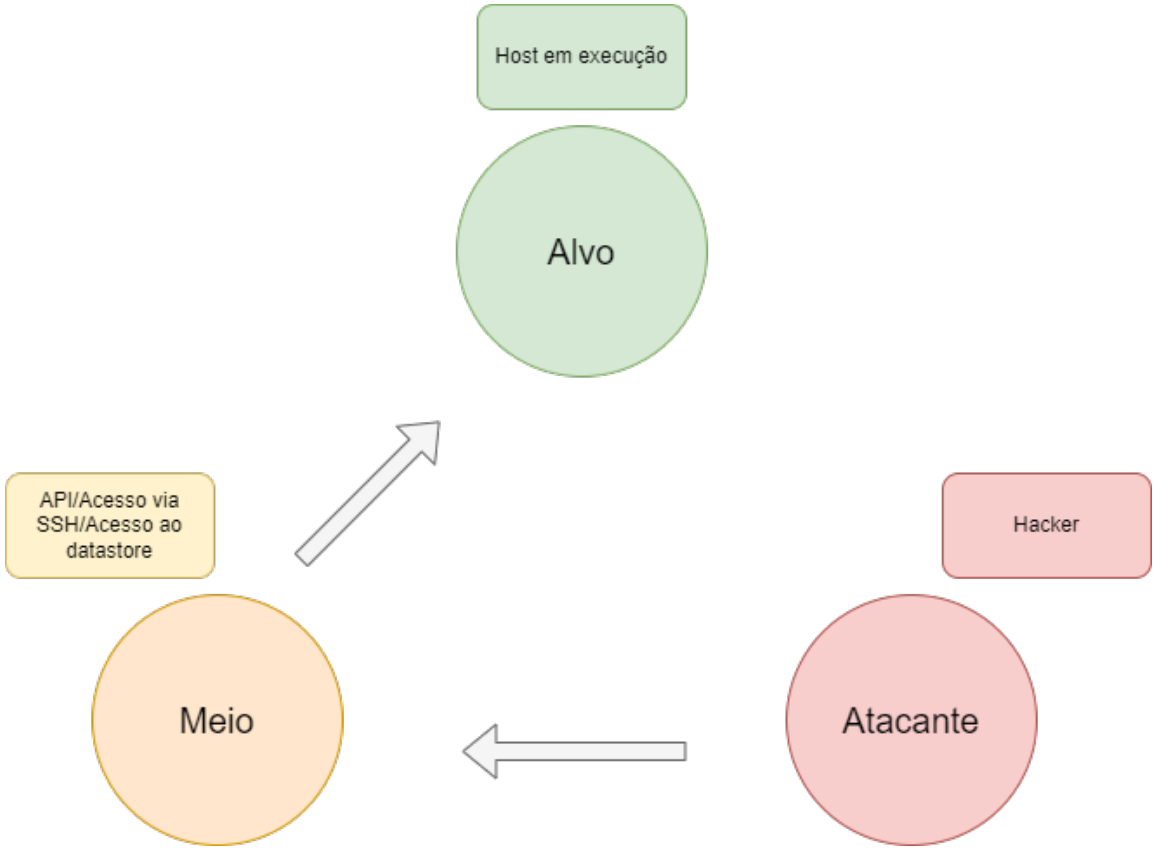


Figura 3.11: Cenário 2 - Processo de Migração - Host

O cenário (3) considera as possíveis vulnerabilidades encontradas no processo de migração de uma máquina virtual partindo da perspectiva da integridade da máquina virtual que está sendo migrada. Nesse cenário, o alvo seria a máquina virtual em execução e o atacante seria um *hacker* com o objetivo de comprometer a máquina virtual que está sendo migrada aproveitando possíveis vulnerabilidades encontradas durante o processo. As vulnerabilidades podem ser derivadas da *API* de comunicação do *cluster*, acesso *SSH* ou acesso

ao *datastore* compartilhado.

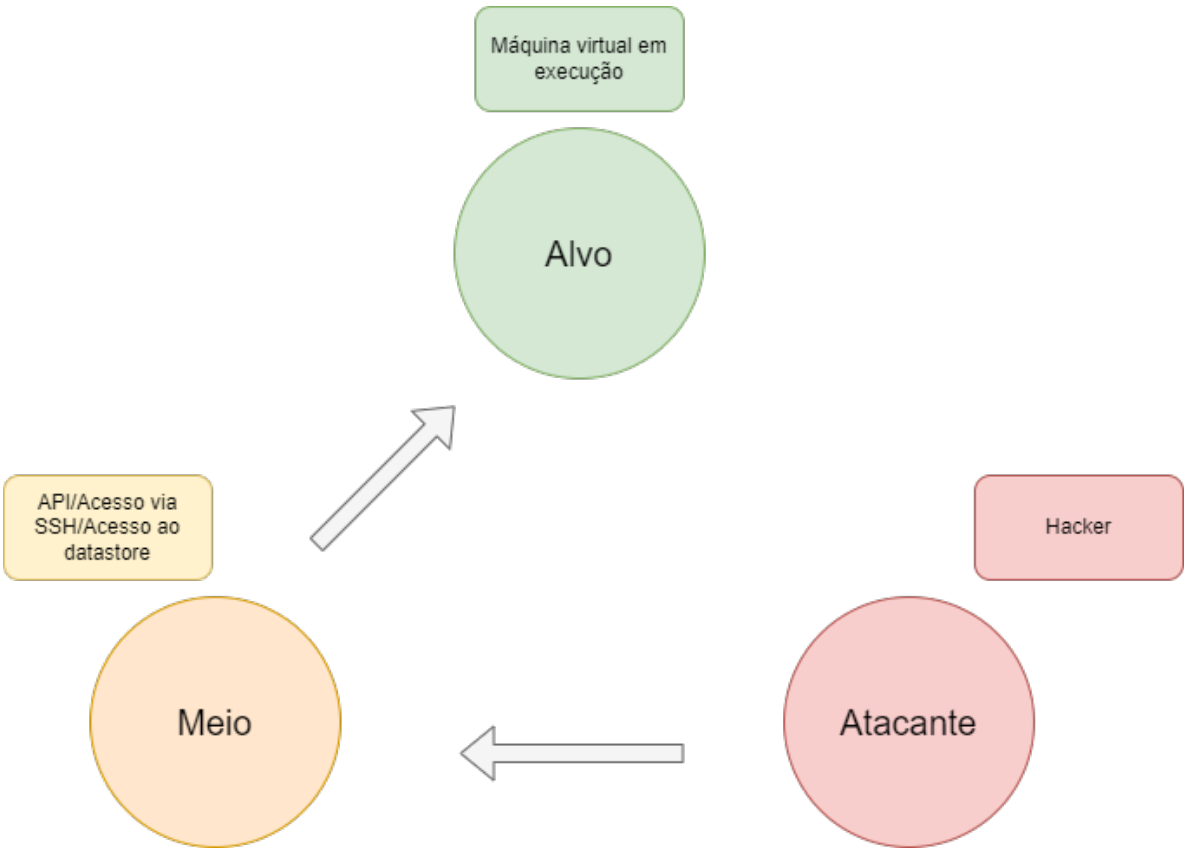


Figura 3.12: Cenário 3 - Processo de Migração - VM

Capítulo 4

Avaliação Experimental e Validação

4.1 Processo de Avaliação Geral

Neste capítulo será exposto o processo para a avaliação da solução proposta. O objetivo da avaliação é comparar o processo de alocação e migração de máquinas virtuais entre os ambientes de virtualização *vSphere* e *KVM*, tendo como clientes os sistemas operacionais Linux Ubuntu e Windows. O processo de avaliação leva em consideração os possíveis parâmetros especificados na tabela 4.3 e as métricas da tabela 4.4. Os cenários de comparação em que as métricas serão avaliadas estão especificados na tabela 4.1, e conta os *inputs* a partir de preferências de um dado administrador, a partir dos possíveis parâmetros de entrada especificados na tabela 4.3.

Neste trabalho também foram realizados experimentos para validar a integridade do sistema e com isso verificar a correta operação do mecanismo proposto nesse trabalho, os detalhes dos cenários de validação de integridade utilizados neste trabalho está especificado na Tabela 4.2. Foram considerados os sistemas operacionais Linux Ubuntu e Windows e

Tabela 4.1: Cenários de comparação para operação.

Cenário	**Virtualização**	**Cliente**
C1	Vsphere	Linux e Windows
C2	Vsphere e KVM	Linux

Tabela 4.2: Cenários de comparação para validação.

Cenário	**Situação**	**Resposta esperada**
CV1	Atestação de TPM falha	Falso
CV2	Atestação de TPM correta, validação de integridade do IMA correta e validação de operações do usuário correta	Íntegro
CV3	Atestação de TPM correta, validação de integridade do IMA correta e validação de operações do usuário falha	Falso
CV4	Atestação de TPM correta, validação de integridade do IMA falha	Falso
CV5	Atestação de TPM correta, validação de integridade do WBCL correta e validação de operações do usuário correta	Íntegro
CV6	Atestação de TPM correta, validação de integridade do WBCL correta e validação de operações do usuário falha	Falso
CV7	Atestação de TPM correta, validação de integridade do WBCL falha	Falso

utilizados os arquivos de medição de integridade *IMA* e *WBCL* para os respectivos sistemas operacionais. Nesta avaliação foram considerando 7 cenários, as situações de ocorrência e os *status* de resposta esperado. Dos cenários especificado existem dois casos em que o status de resposta é **íntegro**, apenas quando a atestação do TPM está **OK**, a validação de integridade do arquivo de Log - IMA para sistemas Linux e WBCL para sistemas Windows - está **OK** e quando a validação das operações do usuário também retorna *status* **OK**.

Tabela 4.3: Parâmetros e entradas para avaliação dos processos.

Parâmetro	Descrição	Entrada
P1	Tecnologia de virtualização	vSphere ou KVM
P2	Requisito de integridade	TPM, vTPM, ambos ou nenhum
P3	Número de requisições (alocação ou migração)	1, 5 ou 10
P4	Sistema operacional	Linux ou Windows

4.1.1 Parâmetros e Métricas

Parâmetros

Para avaliação dos processos propostos foram executados alguns cenários com características específicas. Os parâmetros utilizados estão descritos na tabela 4.3. Dentro da avaliação realizada, foram consideradas dois tipos de tecnologias de virtualização: *vSphere* e *KVM*. Para este experimento foi utilizado um *Cluster* que possui dispositivos com e sem *chip* TPM, assim poderíamos ter como parâmetro a presença ou não do chip, permitindo atender a várias opções do requisito de confiabilidade. Além disso, para avaliar as métricas foram definidos o número de requisições simultâneas no ambiente, especificado no parâmetro P3, que seriam uma, cinco e dez requisições simultâneas, para serem executados tanto para o processo de alocação quanto para o processo de migração de máquinas virtuais. Por fim, também foi observado a variação das métricas de acordo com o sistema operacional em que está sendo executado o experimento, se Windows ou uma distribuição Linux.

Métricas

A Tabela 4.4 especifica as métricas que serão utilizadas para avaliar o processo de execução de alocação e de migração.

- *M1*: Tempo de execução da *task* de alocação ou migração no *Cluster*. Esse tempo considera o momento em que todos os parâmetros e decisões já foram realizadas, e

Tabela 4.4: Métricas utilizada na avaliação experimental do processo.

Métrica	**Descrição**
M1	Tempo de execução do processo
M2	Tempo de decisão do processo
M3	Tempo de recuperação de hosts do cluster
M4	Tempo de recuperação do storage do cluster
M5	Tempo total do processo
M6	Consumo da CPU
M7	Consumo da memória

mede o tempo médio de envio da requisição para o *Cluster* e o retorno *booleano* da tarefa.

- **M2**: Tempo de decisão do processo de alocação ou migração, considerando os parâmetros de entradas de hardware, software e de confiabilidade. Corresponde ao tempo de decisão do melhor *host* e do melhor *storage* (caso tenha sido solicitado armazenamento em *storage* compartilhado).

- **M3**: Essa métrica mede o tempo de solicitação de dados dos *hosts* disponíveis no *cluster*, iniciando no momento em que é solicitado até o momento em que é retornado pela *API*, *SDK* ou *command line* (a depender da implementação).

- **M4**: Essa métrica mede o tempo de solicitação dos dados do *storage* disponíveis para o *Cluster*, iniciando no momento que é solicitado a lista de *datastores* disponíveis até o momento em que é retornado pela *API*, *SDK* ou *command line*.

- **M5**: Essa métrica mede todo o tempo de execução do processo de alocação ou migração. Inicia no início da função de alocação ou migração, e finaliza quando a *task* é retorna um valor *booleano* com o status da operação realizada.

- **M6**: Mede o consumo percentual de *CPU* durante o processo de alocação ou migração. Essa métrica é medida em vários momentos do processo de execução, no início do processo, durante os processos de decisão e recuperação de recursos, e também ao

final da execução. Além do consumo percentual no momento da medição, é capturada também a variação na unidade de medida de memória utilizada por cada ferramenta (Kbits, MB ou GB).

- **M7**: Mede o consumo percentual de memória volátil durante o processo de alocação ou migração. Assim como a **M6**, é medida em vários trechos do processo de execução, variando entre o início, durante as operações de coleta de dados, decisão e ao final do processo.

4.1.2 Ferramentas e Tecnologias Utilizadas

A base do processo proposto foi desenvolvida na linguagem de programação Java e foram utilizadas as seguintes ferramentas e tecnologias nas respectivas versões:

- Eclipse IDE: Version 2021-12 (4.22.0)
- Java JDK 18
- InfluxDB 2.0

Para a integração com os *clusters* específicos como *vSphere* e KVM foram necessários um conjunto de tecnologias e bibliotecas específicas para realizar a comunicação e recuperação de informações dos *clusters*. A depender do *cluster* utilizado será necessário uma forma de comunicação específica, no caso do *ESXi* foi utilizado a plataforma *vSphere* com o gerenciamento do *vCenter* e para o *Cluster KVM* foi desenvolvido uma ferramenta de comunicação limitada aos recursos necessários para este experimento, sendo então projetada para recuperar informações dos *hosts* e suas respectivas máquinas virtuais, bem como realizar os processos de criação e migração de máquinas virtuais dentro dos respectivos *hosts*.

Cluster ESXi

- VMware vSphere SDK: version 7.0.3-19482538
- Bibliotecas complementares
  - Binding provider
  - Javax.ws

  - Pbm

Cluster KVM

- QEMU-kvm
- libvirt
- virt-manager
- libtpms
- SWTPM
- OVMF UEFI

4.1.3 Caracterização dos *Clusters* para a Experimentação

vSphere ESXi

O *cluster vSphere* utilizado na experimentação, é composto de 4 *hosts* físicos com as especificações descritas abaixo.

1. Host1:
    - Versão do Hypervisor: VMWare ESXi, 7.0.3.19482537;
    - Modelo Servidor: ProLiant DL360 Gen10;
    - CPU: 10 núcleos, Intel(R) Xeon(R) Silver 4210R CPU 2.4GHz;
    - RAM: 64GB;
    - Disco: Storage Compartilhado de 4TB.
    - Boot: UEFI;
    - TPM: presente - físico.
2. Host2:
    - Versão do Hypervisor: VMWare ESXi, 7.0.3.17877351;
    - Modelo Servidor: ProLiant DL360 Gen10;

- CPU: 10 núcleos, Intel(R) Xeon(R) Silver 4210R CPU 2.4GHz;
- RAM: 64GB;
- Disco: Storage Compartilhado de 4TB.
- Boot: UEFI;
- TPM: presente - físico.

3. Host3:

- Versão do Hypervisor: VMWare ESXi, 7.0.2.17867351;
- Modelo Servidor: ProLiant BL465c Gen8;
- CPU: 16 núcleos 2.6GHz, AMD Opteron(TM) Processor 6112;
- RAM: 64GB;
- Disco: Storage Compartilhado de 4TB.
- Boot: UEFI;
- TPM: ausente.

4. Host4:

- Versão do Hypervisor: VMWare ESXi, 7.0.2.17867351;
- Modelo Servidor: ProLiant BL465c Gen8;
- CPU: 32 núcleos 2.2GHz, AMD Opteron(TM) Processor 6274;
- RAM: 64GB;
- Disco: Storage Compartilhado de 4TB.
- Boot: UEFI;
- TPM: ausente.

KVM

O *Cluster KVM* foi construído em uma máquina pessoal com processador Core I5, 8 GB de RAM, 240 GB de disco e TPM físico presente. Os *hosts* utilizados no experimento foram virtualizados nessa máquina utilizando o KVM e foram construídos a partir das características especificadas abaixo:

1. Host1:
    - Sistema operacional: Linux;
    - RAM: 2GB;
    - Disco: 50 GB;
    - CPU: 2 núcleos;
    - Boot: UEFI;
    - TPM: presente - virtual.
2. Host2:
    - Sistema operacional: Linux;
    - RAM: 2GB;
    - Disco: 50 GB;
    - CPU: 2 núcleos;
    - Boot: UEFI;
    - TPM: presente - virtual.
3. Host2:
    - Sistema operacional: Linux;
    - RAM: 2GB;
    - Disco: 50 GB;
    - CPU: 2 núcleos;
    - Boot: legacy;
    - TPM: ausente.
4. Host2:
    - Sistema operacional: Linux;
    - RAM: 2GB;

- Disco: 50 GB;
- CPU: 2 núcleos;
- Boot: legacy;
- TPM: ausente.

4.2 Avaliação de Integridade do Sistema

Para a validação da integridade do sistema foram realizadas a avaliação dos dados submetidos ao mecanismo de decisão através de 30 requisições de máquinas reais e virtuais onde em todos os casos o resultado de integridade obtido foi o esperado. Foram realizadas também 5 requisições simulando um estado falho do TPM. Os cenários utilizados na validação de integridade estão especificado na tabela 4.2.

4.2.1 Atestação do TPM falha

CV1 - Atestação de TPM falha

O primeiro cenário avaliado foi o de falha do TPM, como pode ser observado na 4.5. Os *hosts* inicialmente estão com *status* de atestação do TPM **OK** sendo necessário simular um estado de falha na validação. Para isso, os valores de referências das chaves foram alterados e quando os novos valores de chaves dos nós foram submetidos a atestação do TPM dos *hosts* ocorreu a falha. O processo foi replicado para os demais *hosts* do cenário de validação 1.

Tabela 4.5: CV1 - Resultados da execução do experimento no cenário de validação 1.

Nó	Validação do TPM	Resultado Esperado	Resultado Obtido
Host_1	Validado	Falso	Falso
Host_2	Validado	Falso	Falso
Host_3	Validado	Falso	Falso
Host_4	Validado	Falso	Falso
Host_5	Validado	Falso	Falso

4.2.2 Validação de Integridade e Operações de Usuário

CV2 - Atestação de TPM correta, validação de integridade do IMA correta e validação de operações do usuário correta

Para a validação do cenário CV2 assumimos que o TPM foi autenticado e obteve um *status* de validado, assim foram considerados o arquivo do IMA em 5 momentos distintos como pode ser observado na tabela 4.6, os momentos foram nomeados de *Host* de 1 a 5. Esse cenário foi avaliado em duas etapas, primeiro realizou-se uma validação do arquivo de IMA e após da validação com o *status* esperado, o processo prosseguiu para a validação das operações de usuário. Para este cenário considerando o arquivo do *IMA* e a verificação de integridade do arquivo a partir dos valores de referência dos *PCRs* obtidos previamente e comparando os valores de *hashes* obtidos no arquivo do *IMA*.

Realizou-se também uma validação das operações de usuário que consistem basicamente em uma *whitelist* com um conjunto de comando que são permitidos de serem executados dentro do ambiente de validação. Para execução da operação obtivemos o resultado de **validado** para todos os instantes do arquivo do IMA e o resultado de **validado** também para todos os instantes de verificação das operações de usuário, obtendo dessa forma a resposta *íntegro* para todos os *hosts* avaliados durante o CV2.

Tabela 4.6: CV2 - Resultados da execução do experimento no cenário de validação 2.

Nó	**Validação do IMA**	**Validação Operações de Usuário**	**Resultado Esperado**	**Resultado Obtido**
Host_1	Validado	Falso	Falha	Falha
Host_2	Validado	Falso	Falha	Falha
Host_3	Validado	Falso	Falha	Falha
Host_4	Validado	Falso	Falha	Falha
Host_5	Validado	Falso	Falha	Falha

CV3 - Atestação de TPM correta, validação de integridade do IMA correta e validação de operações do usuário falha

Para a validação do cenário CV3 assumimos que o TPM foi autenticado e obteve um *status* de validado, assim foram considerados o arquivo do IMA em 5 momentos distintos, como pode ser observado na tabela 4.7, os momentos foram nomeados de *Host* de 1 a 5. Esse cenário foi avaliado em duas etapas, primeiro realizou-se uma validação do arquivo de IMA e após da validação com o *status* esperado, o processo prosseguiu para a validação das operações de usuário. Para este cenário foram considerados o arquivo do *IMA* para verificação de integridade do arquivo a partir dos valores de referência dos *PCRs* obtidos previamente e comparando os valores de *hashes* obtidos no arquivo do *IMA*.

Realizou-se também uma validação das operações de usuário que consistem basicamente em uma *whitelist* com um conjunto de comandos que são permitidos de serem executados dentro do ambiente de validação. Para execução da operação obtivemos o resultado de **validado** para todos os instantes do arquivo do IMA e o resultado de **falso** também para todos os instantes de verificação das operações de usuário, obtendo dessa forma a resposta *falha* na validação de integridade do CV3. Neste cenário, como a validação das operações de usuário ocorreu com falha a validação de integridade geral também resulta em falha, uma que vez a validação de integridade nesse ponto é considerada em conjunto com a validação do IMA e da *whitelist*.

Tabela 4.7: CV3 - Resultados da execução do experimento no cenário de validação 3.

Nó	**Validação do IMA**	**Validação Operações de Usuário**	**Resultado Esperado**	**Resultado Obtido**
Host_1	Validado	Validado	Íntegro	Íntegro
Host_2	Validado	Validado	Íntegro	Íntegro
Host_3	Validado	Validado	Íntegro	Íntegro
Host_4	Validado	Validado	Íntegro	Íntegro
Host_5	Validado	Validado	Íntegro	Íntegro

CV4 - Atestação de TPM correta, validação de integridade do IMA falha

Para a validação do cenário CV4 assumimos que o TPM foi autenticado e obteve um *status* de validado, assim foram considerados o arquivo do IMA em 5 momentos distintos, como pode ser observado na tabela 4.8, os momentos foram nomeados de *Host* de 1 a 5. Esse cenário foi avaliado em uma etapa considerando a validação do arquivo de IMA. Para este cenário a validação do IMA ocorre a partir dos valores de referência dos *PCRs* obtidos previamente e comparando com os valores de *hashes* obtidos no arquivo do *IMA*.

Neste cenário de execução obtivemos o *status* de atestação **falso** para a validação do arquivo de IMA, como o *host* não passou desta etapa com êxito não prossegue para a validação das operações de usuário, e retorna uma resposta de falha no processo geral de validação de integridade.

Tabela 4.8: CV4 - Resultados da execução do experimento no cenário de validação 4.

Nó	Validação do IMA	Resultado Esperado	Resultado Obtido
Host_1	Falso	Falha	Falha
Host_2	Falso	Falha	Falha
Host_3	Falso	Falha	Falha
Host_4	Falso	Falha	Falha
Host_5	Falso	Falha	Falha

CV5 - Atestação de TPM correta, validação de integridade do WBCL correta e validação de operações do usuário correta

Para a validação do cenário CV5 assumimos que o TPM foi autenticado e obteve um *status* de validado, assim foram considerados o arquivo do WBCL em 5 momentos distintos como pode ser observado na tabela 4.9, os momentos foram nomeados de *Host* de 1 a 5. Esse cenário foi avaliado em duas etapas, primeiro realizou-se uma validação do arquivo de WBCL e após da validação com o *status* esperado, o processo prosseguiu para a validação das operações de usuário. Para este cenário considerando o arquivo do *WBCL* e a verificação de integridade do arquivo a partir dos valores de referência dos *PCRs* obtidos previamente e

comparando os valores de *hashes* obtidos no arquivo do *WBCL.*

Realizou-se também uma validação das operações de usuário que consistem basicamente em uma *whitelist* com um conjunto de comando que são permitidos de serem executados dentro do ambiente de validação. Para execução da operação obtivemos o resultado de **validado** para todos os instantes do arquivo do WBCL e o resultado de **validado** também para todos os instantes de verificação das operações de usuário, obtendo dessa forma a resposta *íntegro* para todos os *hosts* avaliados durante o CV5.

Tabela 4.9: CV5 - Resultados da execução do experimento no cenário de validação 5.

Nó	**Validação do WBCL**	**Validação Operações de Usuário**	**Resultado Esperado**	**Resultado Obtido**
Host_1	Validado	Validado	Íntegro	Íntegro
Host_2	Validado	Validado	Íntegro	Íntegro
Host_3	Validado	Validado	Íntegro	Íntegro
Host_4	Validado	Validado	Íntegro	Íntegro
Host_5	Validado	Validado	Íntegro	Íntegro

CV6 - Atestação de TPM correta, validação de integridade do WBCL correta e validação de operações do usuário falha

Para a validação do cenário CV3 assumimos que o TPM foi autenticado e obteve um *status* de validado, assim foram considerados o arquivo do WBCL em 5 momentos distintos, como pode ser observado na tabela 4.10, os momentos foram nomeados de *Host* de 1 a 5. Esse cenário foi avaliado em duas etapas, primeiro realizou-se uma validação do arquivo de WBCL e após da validação com o *status* esperado, o processo prosseguiu para a validação das operações de usuário. Para este cenário foram considerados o arquivo do *WBCL* para verificação de integridade do arquivo a partir dos valores de referência dos *PCRs* obtidos previamente e comparando os valores de *hashes* obtidos no arquivo do *WBCL.*

Realizou-se também uma validação das operações de usuário que consistem basicamente em uma *whitelist* com um conjunto de comandos que são permitidos de serem executados

dentro do ambiente de validação. Para execução da operação obtivemos o resultado de **validado** para todos os instantes do arquivo do WBCL e o resultado de **falso** também para todos os instantes de verificação das operações de usuário, obtendo dessa forma a resposta *falha* na validação de integridade do CV6. Neste cenário, como a validação das operações de usuário ocorreu com falha a validação de integridade geral também resulta em falha, uma que vez a validação de integridade nesse ponto é considerada em conjunto com a validação do WBCL e da *whitelist*.

Tabela 4.10: CV6 - Resultados da execução do experimento no cenário de validação 6.

Nó	**Validação do WBCL**	**Validação Operações de Usuário**	**Resultado Esperado**	**Resultado Obtido**
Host_1	Validado	Falso	Falha	Falha
Host_2	Validado	Falso	Falha	Falha
Host_3	Validado	Falso	Falha	Falha
Host_4	Validado	Falso	Falha	Falha
Host_5	Validado	Falso	Falha	Falha

CV7 - Atestação de TPM correta, validação de integridade do WBCL falha

Para a validação do cenário CV7 assumimos que o TPM foi autenticado e obteve um *status* de validado, assim foram considerados o arquivo do IMA em 5 momentos distintos, como pode ser observado na tabela 4.11, os momentos foram nomeados de *Host* de 1 a 5. Esse cenário foi avaliado em uma etapa considerando a validação do arquivo de WBCL. Para este cenário a validação do WBCL ocorre a partir dos valores de referência dos *PCRs* obtidos previamente e comparando com os valores de *hashes* obtidos no arquivo do *WBCL*.

Neste cenário de execução obtivemos o *status* de atestação **falso** para a validação do arquivo de WBCL, como o *host* não passou desta etapa com êxito não prossegue para a validação das operações de usuário, e retorna uma resposta de falha no processo geral de validação de integridade.

Tabela 4.11: CV7 - Resultados da execução do experimento no cenário de validação 7.

Nó	**Validação do WBCL**	**Resultado Esperado**	**Resultado Obtido**
Host_1	Falso	Falha	Falha
Host_2	Falso	Falha	Falha
Host_3	Falso	Falha	Falha
Host_4	Falso	Falha	Falha
Host_5	Falso	Falha	Falha

4.3 Avaliação do Processo de Alocação

Para a avaliação do processo de alocação, executamos alguns cenários de testes considerando os parâmetros de entrada especificados na tabela 4.3. As métricas foram avaliadas sob a perspectiva de dois grupos: Grupo 1 - Tempo de execução (tempo de *request* do processo, tempo de decisão, tempo de recuperação de hosts, tempo de recuperação de *storage* e tempo total do processo); e Grupo 2 - consumo de recursos (consumo de CPU e consumo de memória).

4.3.1 Tempo de Execução

M1 - Tempo de *request* da Alocação

O tempo de *request* da alocação reflete o tempo médio entre a diferença do tempo de chegada de todos os parâmetros na função que irá realizar a operação direta no *Cluster* e o retorno *booleano* dessa função. Essa métrica irá demonstrar o tempo médio do *Cluster* para realizar a operação, independentemente de outras etapas que estarão sendo realizadas, como recuperação de recursos ou decisão de *hosts*, por exemplo.

A métrica M1 foi avaliada em dois cenários, conforme demonstrado na tabela 4.1. A figura 4.1 mostra uma comparação para os ambiente *vSphere* com clientes Windows e Linux, a fim de validar os resultados e comparar os cenários dos ambientes criados. Nesta figura, conseguimos perceber que no ambiente Linux há uma maior variabilidade entre as médias coletadas nas iterações, os dados estão mais consistentes na distribuição entre quartis

e entre os valores máximos e mínimos. No ambiente Windows, pode-se perceber uma maior presença de *outliers*, mas uma menor variabilidade no conjunto de dados.

Na métrica M1, em termos absolutos, é possível perceber que o tempo médio do ambiente Windows para execução do processo de *request* tende a ser um pouco maior quando temos uma quantidade menor de requisições, mas apresenta menores valores ao aumentarmos a quantidade de requisições quando comparado ao ambiente em Linux. Essa diferença deve ocorrer devido às diferenças de bibliotecas necessárias para cada um dos ambientes.

Uma característica para ambos os ambientes é a tendência de redução no valor do tempo ao aumentarmos a quantidade de requisições. Isso ocorre devido a mecanismos de caches de consultas das informações do *Cluster* que são utilizados pelo *vSphere*, permitindo que consultas subsequentes sejam mais rápidas.

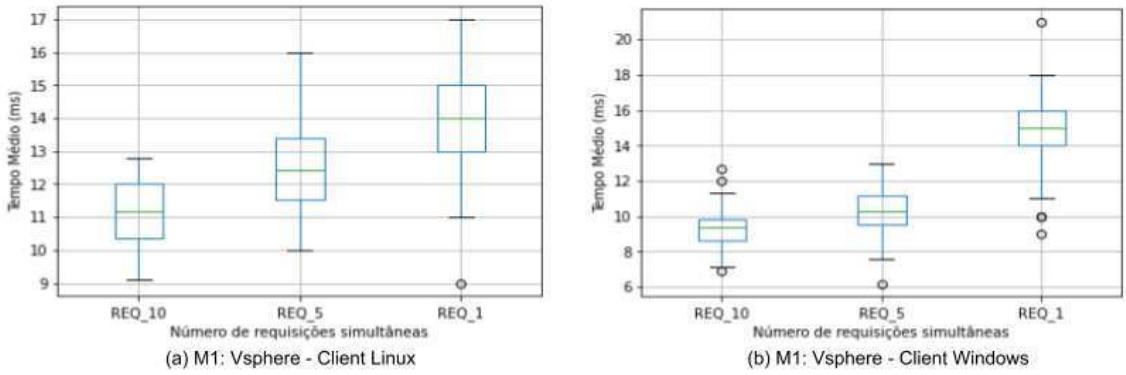


Figura 4.1: Cenário 1 - M1: vSphere - Cliente Windows e Cliente Linux

Dentro dessa mesma métrica, há o cenário 2 que realiza uma comparação da M1 nos ambientes *vSphere* e KVM, ambos com clientes Linux. A partir da figura 4.2, podemos observar em ambos os ambientes que há uma tendência de que iterações com apenas uma requisição simultânea apresentem um tempo médio superior às iterações com cinco e dez requisições simultâneas. Observa-se também uma maior variabilidade do conjunto de dados no ambiente *vSphere*, enquanto que no ambiente KVM há uma maior presença de pontos discrepantes que foram de máximos e mínimos comuns, presentes principalmente para as iterações com uma e cinco requisições simultâneas. Assim, como no cenário anterior, as variações relacionadas a diminuição do tempo médio ao aumentar o número de requisições

simultâneas deve ocorrer devido à um mecanismo de cache das tecnologias.

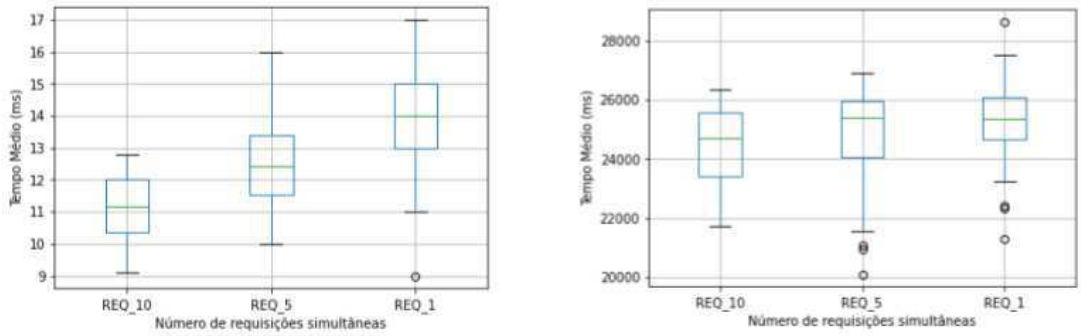


Figura 4.2: Cenário 2 - M1: vSphere e KVM - Cliente Linux

M2 - Tempo de Decisão

O tempo de decisão reflete a diferença entre o tempo de solicitação dos dados dos *hosts* e dos *storages* e o tempo em que são retornados pelo *cluster* para uso da ferramenta. A utilização dessa medição como métrica da avaliação tem o objetivo de observar a variação do tempo de decisão da ferramenta para escolha do *host* para o processo de alocação. A figura 4.3 mostra a variação do tempo de decisão dentro do ambiente de virtualização *vSphere*, utilizando Windows e Linux como sistemas operacionais clientes para as máquinas hospedeiras. Como é possível observar os dois clientes tendem a ter um comportamento semelhante quanto ao aumento do tempo médio desproporcional à quantidade de requisições, na figura onde quanto menor o tempo de utilização do *Cluster* para cada iteração maior o tempo médio.

Com base nisso, que as variações dos tempos médios coletados devem ocorrer devido às alterações ocasionadas pelo modelo de implementação interna do *Cluster*. No *vSphere*, há uma tendência em observar que os dados solicitados e frequentemente utilizados ficam em uma espécie de cache para utilização nas demais iterações.

Quando observamos a figura 4.4, percebe-se de forma mais clara a interferência do ambiente. Na figura 4.4 (a) temos o cliente Linux no *vSphere*, e na figura 4.4 (b) observamos o mesmo tipo de cliente no ambiente de virtualização do KVM. Percebemos que o *vSphere* mantém a relação observada em M1, quanto a variação do tempo médio em decorrência da

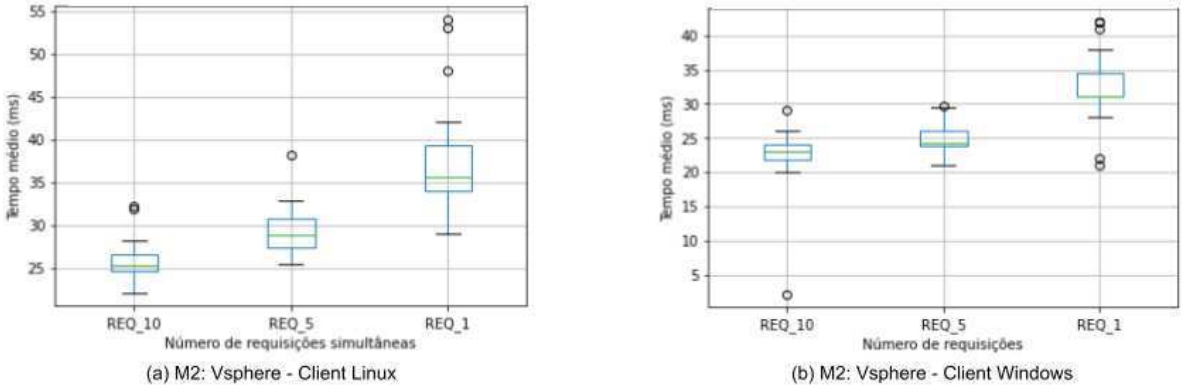


Figura 4.3: Cenário 1 - M2: vSphere - Cliente Windows e Cliente Linux

quantidade de requisições simultâneas, e no ambiente KVM, para a métrica M2 observa-se que o aumento no número de requisições interfere diretamente no aumento do tempo médio destinado ao processo decisório, demonstrando não seguir a relação de consumo e armazenamento de recurso do ambiente *vSphere ESXi*.

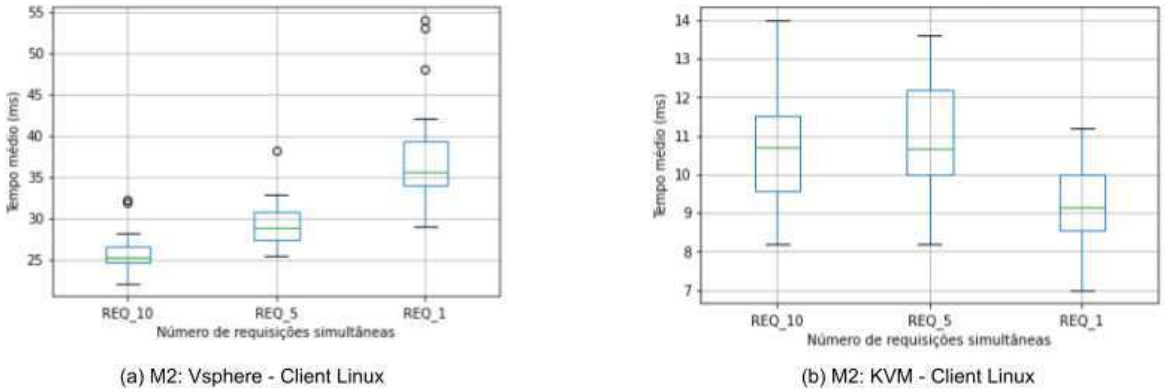


Figura 4.4: Cenário 2 - M2: vSphere e KVM - Cliente Linux

M3 - Tempo de recuperação de hosts do Cluster

O tempo de recuperação de *hosts* reflete a diferença entre o tempo de requisição da lista de *hosts* do *Cluster* e o instante em que são retornados para o componente de decisão. A figura 4.5 exibe uma comparação entre os clientes Linux e Windows no ambiente *vSphere*.

Na figura, observa-se a amplitude dos dados para ambos os ambientes, na *REQ_1*, a amplitude é maior quando comparada com as observações de *REQ_5* e *REQ_10*. Vale observar também que a comparação em termos absolutos nos dois clientes permite afirmar, apesar de ambos terem a mesma tendência no comportamento, que o ambiente Linux consome mais tempo no processo de recuperação que o ambiente Windows, isso acontece devido à fatores de uma maior compatibilidade do ambiente Windows com as tecnologias *vSphere ESXi*. A característica de cache destacada nas discussões anteriores torna-se mais evidente na discussão desta métrica, como podemos observar para 5 e 10 requisições simultâneas, temos uma baixa amplitude dos dados e uma baixa variação interquartílica, o que nos permitir concluir que as variações nas iterações tendem a apresentar um comportamento similar a depender do cliente utilizado.

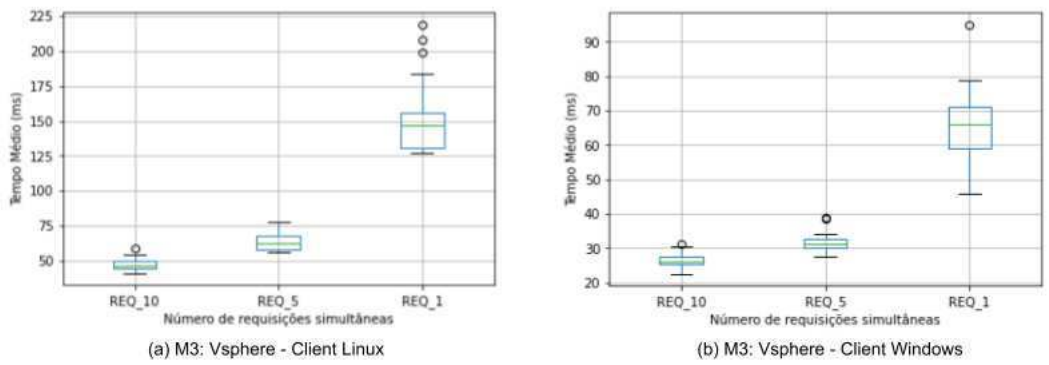


Figura 4.5: Cenário 1 - M3: vSphere - Cliente Linux e Windows

Aa comparar o cliente Linux nos ambientes *vSphere* e KVM na Figura 4.6 (a), observa-se o mesmo padrão apresentado no cenário anterior, mas 4.6 (b), mostra a mesma tendência observada em M2, apresentando uma alta amplitude para 5 e 10 requisições, e uma alta variação interquarlítica. A comparação em termos absolutos não pode ser realizada dada as características de hardware muito distintas dos dois ambientes, mas para essa métrica (M3), os dois ambientes apresentaram comportamentos diferentes, essa variação deve ocorrer devido a característica de cache do ambiente *vSphere*, em que para as métricas analisadas até o momento pode-se encontrar um comportamento semelhante fortalecendo a ideia levantada no início da análise.

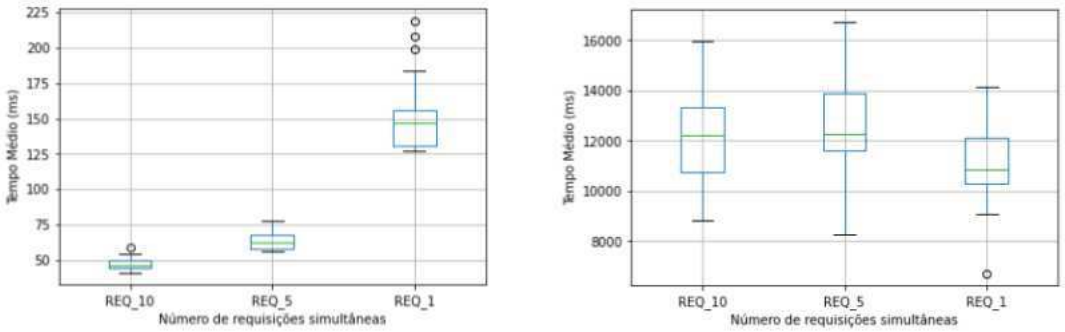


Figura 4.6: Cenário 2 - M3: vSphere e KVM - Cliente Linux

M4 - Tempo de recuperação do *storage* do Cluster

O tempo de recuperação dos *storages* representa uma métrica complementar utilizada para comparar os clientes Linux e Windows no ambiente vSphere. Por esse ambiente apresentar a utilização de mais de uma unidade de armazenamento compartilhado, foi possível escolher aquele que tivesse a maior capacidade de disco disponível para alocação da máquina virtual. A figura 4.7 mostra os resultados do dados coletados com a diferença entre o tempo de requisição e o tempo de retorno da lista de *storages* do *Cluster vSphere*.

Observa-se na figura 4.7 que há uma tendência no aumento do tempo de permanência em conexão ativa no *Cluster* influenciar no tempo médio para recuperação dos dados à medida em que há o aumento no número de requisições simultâneas que serão disparadas. A comunicação do Linux com o ambiente apresenta uma maior presença de *outliers*, mas mantém uma baixa amplitude dos dados, enquanto que no ambiente Windows há uma maior amplitude entre os dados coletados. Além disso, é possível observar na REQ_1 um comportamento atípico onde não tem-se um limite superior definido e a maioria dos dados estão concentrados acima da mediana. Novamente, comparando os dois clientes no *vSphere*, percebemos que o cliente Windows apresenta um tempo médio de recuperação mais rápido que o ambiente Linux, isso deve acontecer relacionado ao fator de maior compatibilidade do sistema operacional Windows com as tecnologias do *vSphere*.

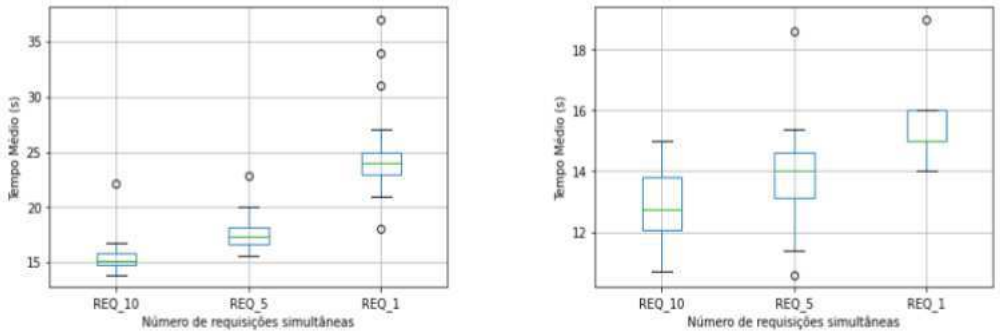


Figura 4.7: Cenário 1 - M4: vSphere - Cliente Linux e Windows

M5 - Tempo total do processo

O tempo total do processo representa a diferença entre o tempo de entrada na função de alocação até o retorno da função com o status de execução. Essa métrica considera o tempo de recuperação de *hosts*, *storages* (quando for o caso), tempo de decisão e o tempo do processo de *request* da operação no *cluster*. A figura 4.8 exibe a comparação entre Windows e Linux no ambiente vSphere, é possível observar que em termos absolutos o cliente Windows tem um consumo médio de tempo menor que o cliente Linux em todos os cenários de requisições, mas para o REQ_1 observa-se um comportamento atípico com a ausência de limite inferior. Nas demais requisições, tem-se a mediana muito próxima ao primeiro quartil demonstrando uma assimetria positiva, apesar de 50% dos dados estarem acima do Q2. No ambiente Linux, observa-se uma distinção muito expressiva entre a REQ_1 e as demais requisições, apresentando tanto um maior tempo médio total, quanto uma maior amplitude dos dados. Em termos tendenciais, ambos os clientes apresentaram comportamentos semelhantes, com tendência a diminuição do tempo total à medida em há um aumento no número de requisições simultâneas, mostrando que para esta métrica a característica de cache observada nas métricas anteriores também está presente.

Ao observar um comparativo entre o *vSphere* e o KVM com o cliente Linux, pode-se observar que o ambiente KVM na métrica M5, mostra um comportamento tendencial semelhante ao ambiente *vSphere*. Diferentemente de outras métricas analisadas, o KVM tende a aumentar o tempo total da alocação à medida em que há um aumento no número de re-

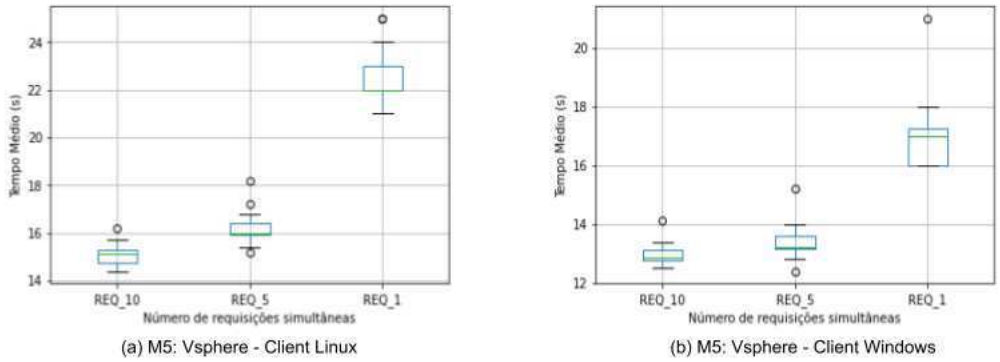


Figura 4.8: Cenário 1 - M5: vSphere - Cliente Linux e Windows

quisições simultâneas. Nas métricas anteriores analisadas do KVM, observamos que este ambiente de virtualização tinha um comportamento distinto do *vSphere* em relação as métricas de tempo quando analisado o número de requisições disparadas, mas nesta métrica em específico, observa-se um comportamento semelhante em ambos os ambientes.

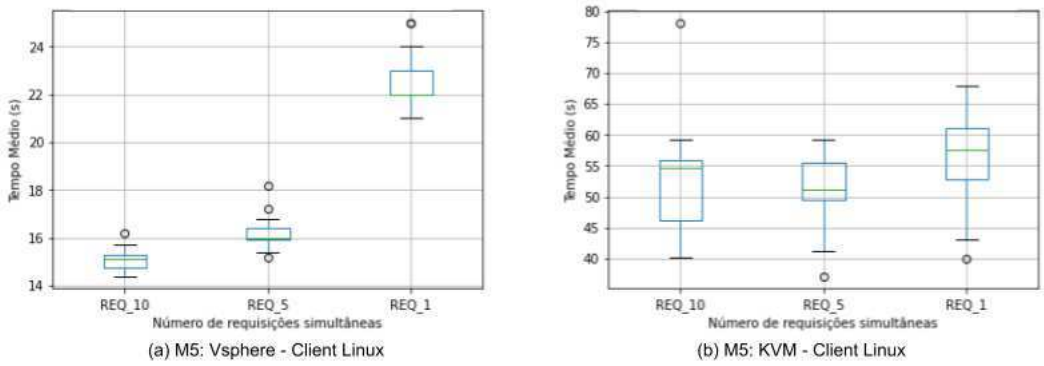


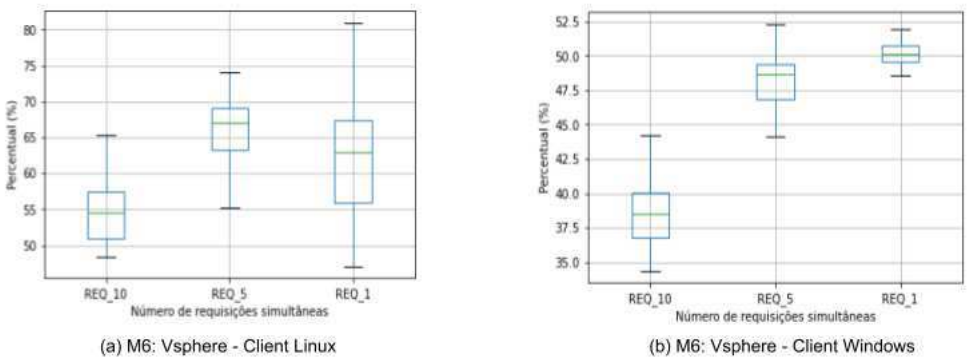
Figura 4.9: Cenário 2 - M5: vSphere e KVM - Cliente Linux

4.3.2 Consumo de Recursos

M6 - Consumo de CPU

A métrica M6 objetiva medir o consumo percentual de CPU durante o processo de alocação. São realizadas medições em vários instantes do processo de execução. A figura 4.10 mostra

um comparativo no ambiente *vSphere* com os clientes Linux e Windows, como pode-se observar nas variáveis REQ_10 e REQ_5, ambos os ambientes mantém um padrão no tempo médio em relação ao aumento do número de requisições. A variável REQ_1 do cliente Linux apresentou um comportamento diferente, tendo sua amplitude atingindo os valores máximo e mínimo possíveis, isso se deve ao fato de uma possível instabilidade na comunicação do cliente Linux com o *cluster vSphere*, em todas as observações de consumo de CPU, foi possível notar a presença dessa variação nas medições. Além disso, nota-se um intervalo interquartílico muito superior quando comparado com a mesma variável do ambiente Windows. Em termos de valores absolutos, o ambiente com cliente Windows tende a apresentar melhores resultados em todas as variáveis de requisições apresentadas, corroborando a ideia de melhor compatibilidade do ambiente Windows com as tecnologias *vSphere*.



(a) M6: Vsphere - Client Linux

(b) M6: Vsphere - Client Windows

Figura 4.10: Cenário 1 - M6: vSphere - Cliente Linux e Windows

Ao analisar a comparação entre os clientes Linux nos ambientes KVM e *vSphere* na figura 4.11 observa-se um comportamento diferente, tanto em termos tendenciais quanto nos valores absolutos. Enquanto que no ambiente *vSphere*, uma das variáveis atingiu os valores máximos e mínimos, no KVM os dados mantiveram-se mais estáveis, apresentando um menor tempo médio para a *REQ_5* e intervalos semelhantes para 1 e 10 requisições.

M7 - Consumo de Memória

Esta métrica, assim como a anterior, objetiva medir em vários instantes do processo de execução da alocação o consumo de memória, realizando as medições em vários instantes da

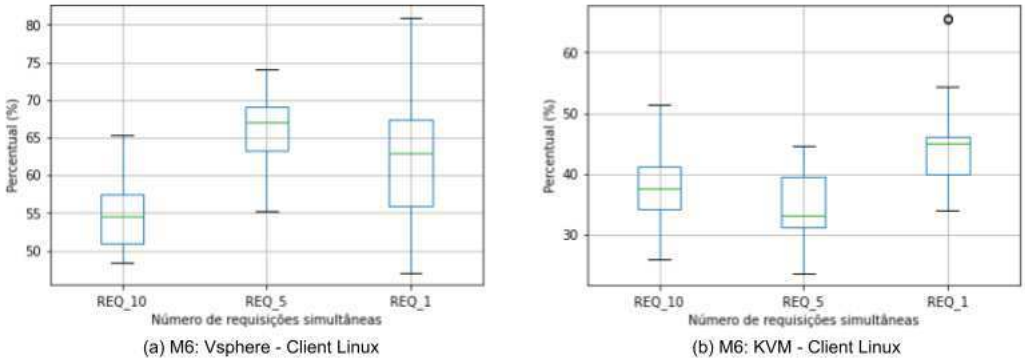


Figura 4.11: Cenário 2 - M6: vSphere e KVM - Cliente Linux

execução. A figura 4.12 exibe a comparação entre os clientes Linux e Windows no ambiente *vSphere*, ambos os ambientes apresentam o mesmo comportamento tendencial e diferente de todas as observações anteriores. Nessa métrica, quanto maior o número de requisições simultâneas, maior o percentual médio de consumo de memória, e em termos absolutos, o desempenho do cliente Linux foi relativamente superior ao Windows. Isso se deve à maior compatibilidade do cliente Windows com as tecnologias do ambiente *vSphere*.

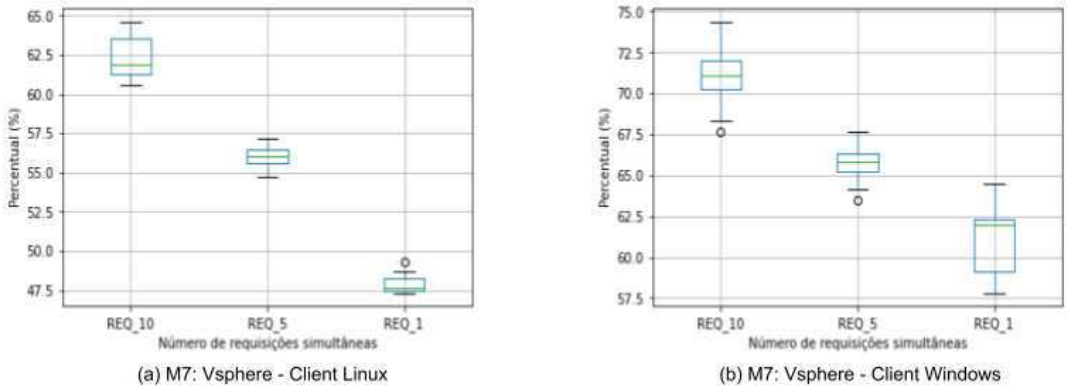


Figura 4.12: Cenário 1 - M7: vSphere - Cliente Linux e Windows

Ao comparar os ambientes *vSphere* e KVM, pode-se notar que para as variáveis REQ_10 e REQ_5 há um comportamento semelhante, mas a variável REQ_1 do ambiente KVM apresenta um comportamento distinto, tendo ausência do limite inferior e possuindo a mediana estritamente próximo ao Q1, tendo os dados acima do valor da mediana. Como observado na

métrica anterior, a coleta dos percentuais de consumo de CPU e memória no cliente Linux apresentou maiores inconsistências com as observações das métricas anteriores, o que pode está relacionado com as restrições do ambiente de execução e com as tecnologias utilizada para comunicação com os ambientes.

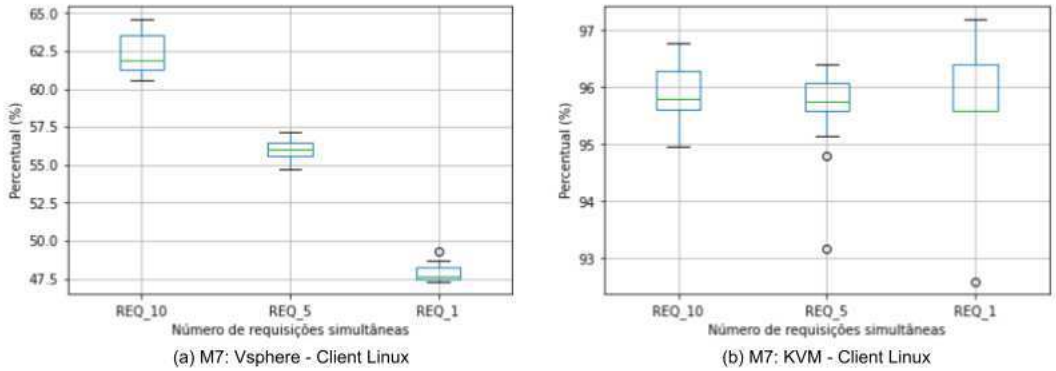


Figura 4.13: Cenário 2 - M7: vSphere e KVM - Cliente Linux

Validação

A partir das análises das métricas para o processo de alocação, observou-se diferentes comportamentos para os ambientes de virtualização *vSphere ESXi* e *KVM*, e também para os clientes Windows e Linux utilizados na experimentação. Como pôde-se notar ao longo do detalhamento dos resultados, há algumas variáveis que possui uma relação tendencial maior do que outras. A figura 4.14 mostra a correlação numérica entre as variáveis do ambiente de virtualização KVM com cliente Linux. A variação ocorre entre o intervalo [1, -1], onde 1 representa uma correlação forte e positiva, -1 representa uma correlação forte e negativa e 0 representa valores não correlacionadas. Valores positivos e mais próximos a zero representam uma correlação fraca e positiva, enquanto valores negativos e mais próximos de zero representam uma correlação fraca e negativa.

Para a análise da correlação, estabelecemos três margens variando em 25%, 50% e 75% de correlação entre as variáveis, iniciando de forma decrescente, das variáveis com maior correlação para as de menor correlação. Para observação do ambiente KVM Linux, observou-se que não há nenhum par com correlação acima de 50% ou 75%, dessa forma, foi

estabelecido a margem percentual de observação de 25% de correlação. Com essa margem, observou-se uma correlação para os pares *1 - [Requisicao, Time_Decision]*, *2 - [Requisicao, Consume_CPU]*, apresentando uma correlação igual ou inferior a margem. A correlação entre essas variáveis demonstram uma relação forte. No primeiro par, quando há um aumento no número de requisições há uma aumento no tempo médio para realizar a operação. No segundo par, há um relação inversa, quando há um aumento do número de requisições simultâneas, o consumo percentual médio de CPU foi diminuindo, o que pode ter sido ocasionado devido ao cache de dados durante a execução do processo, mas seria necessário uma amostra maior e a coleta de mais dados para confirmação da relação.

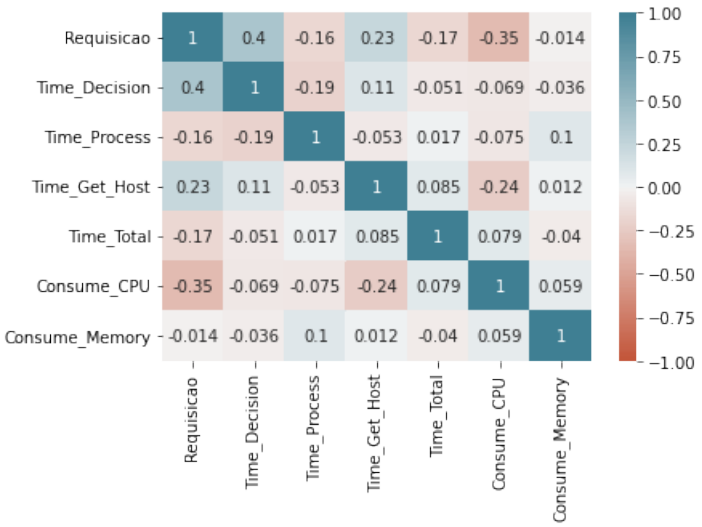


Figura 4.14: KVM Linux: Correlação entre as métricas do ambiente.

Ao analisar o ambiente *vSphere* Windows na Figura 4.15, com a margem de correlação em 75%, pode-se observar que existem várias variáveis com uma forte relação. Nos pares destacados abaixo, é possível perceber uma relação positiva entre o consumo de memória e o número de requisições simultâneas, e do tempo total com o tempo do processo. As demais relações destacadas com sinal negativo pode indicar a influência do cache dos dados da tecnologia de virtualização, quando se trata do aumento do número de requisições simultâneas.

1. Requisicao e Time_Total = forte negativa,

2. Requisicao e Consume_CPU = forte negativa,

3. Requisicao e Consume_Memory = forte positiva,

4. Time_Process e Time_Total = forte positiva,

5. Time_Total e Consume_Memory = forte negativa,

6. Consume_CPU e Consume_Memory = forte negativa.

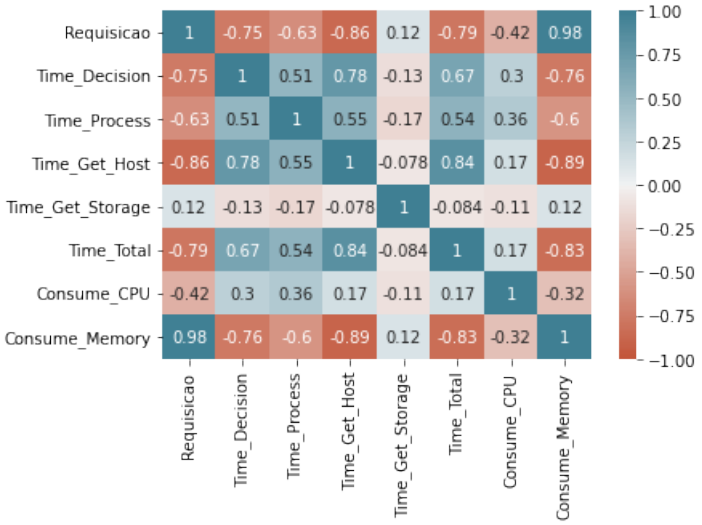


Figura 4.15: vSphere Windows: Correlação entre as métricas do ambiente.

No mapa da figura 4.16, com as correlações entre as variáveis do ambiente *vSphere* Linux, é possível destacar a presença de mais relações fortes e positivas entre as variáveis, mantendo um comportamento semelhante ao ambiente anterior.

A partir dos dados de correlação entre as variáveis do processo de alocação, foi possível perceber que há uma forte relação positiva entre o consumo de memória e o tempo total, como também entre o consumo de memória e o número de requisições simultâneas, demonstrando que o aumento em uma métrica interfere positivamente no crescimento da outra. Além disso, há uma forte tendência em considerar a existência de cache dentro dos ambientes de virtualização, permitindo com que determinadas dados recuperados no ambiente permaneçam cache por um determinado período de tempo, demonstrando a correlação forte negativa, por exemplo, entre o consumo de CPU e o consumo de memória.

De modo geral, há distinções entre os ambientes de virtualização, a partir a observação dos tempos médios destaca-se o maior desempenho do ambiente *vSphere* Windows na

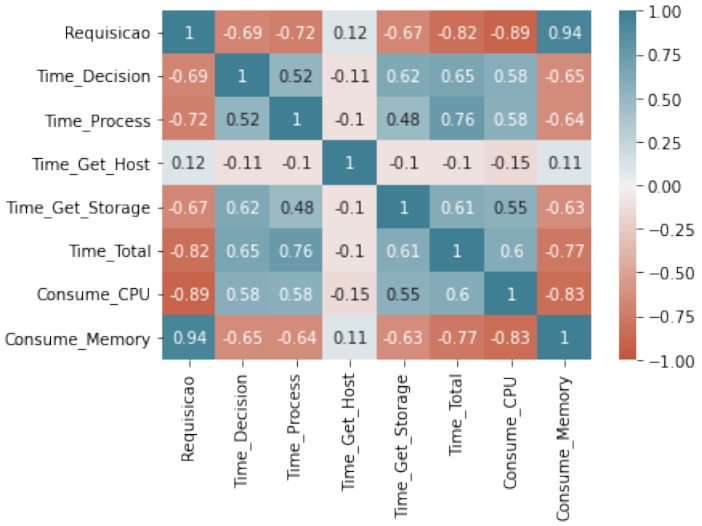


Figura 4.16: vSphere Linux: Correlação entre as métricas do ambiente.

execução das operações, como mostrado nas discussões anteriores, isso ocorre devido a maior compatibilidade do cliente Windows com as tecnologias e dependências do ambiente *vSphere*, que fornece maior suporte e disponibilidade de recursos através de documentações mais atualizadas, e diferentes versões de *API* e *SDKs* para ambientes compatíveis com Windows.

4.4 Avaliação do Processo de Migração

Para a avaliação do processo de migração, executamos alguns cenários de testes considerando os parâmetros de entrada especificados na tabela 4.3. As métricas avaliadas nesta seção serão as mesmas que foram avaliadas na seção de avaliação do processo de alocação. O processo de migração consiste na migração de máquinas virtuais de um *host* que perdeu o seu status de confiabilidade ou que deixou de atender os requisitos de software ou hardware especificados no momento de criação da máquina virtual.

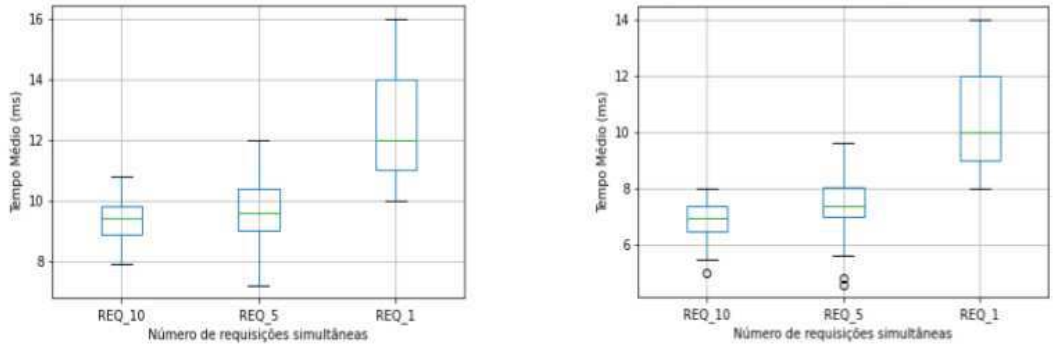
A avaliação do processo de migração levou em consideração um ambiente em que ocorreu 1, 5 e 10 requisições simultâneas de máquinas virtuais para serem migradas entre *hosts*. As métricas foram avaliadas sob a perspectiva de dois grupos: Grupo 1 - Tempo de execução (tempo de *request* do processo, tempo de decisão, tempo de recuperação de *hosts*, tempo de recuperação de *storage* - quando for o caso, e tempo total do processo); e Grupo 2 - consumo

de recursos (consumo de CPU e consumo de memória).

4.4.1 Tempo de Execução

M1 - Tempo de *request* da Migração

O tempo de *request* do processo de migração reflete a diferença do tempo de chegada de todos os parâmetros na função que irá realizar a operação direta no *Cluster* e o retorno *booleano* dessa função. Essa métrica irá demonstrar o tempo médio do *cluster* para realizar a operação, independentemente de outras etapas que estarão sendo realizadas, como recuperação de recursos ou decisão de *hosts*, por exemplo. A figura 4.17 exibe um comparativo entre os clientes Linux e Windows no ambiente *vSphere*. É possível observar que em ambos os ambientes há a mesma tendência nos dados, com amplitude e intervalos interquartílicos semelhantes, em valores absolutos o ambiente Windows apresenta menor tempo médio para realização da operação.



(a) M1: Vsphere - Client Linux

(b) M1: Vsphere - Client Windows

Figura 4.17: Cenário 1 - M1: vSphere - Cliente Windows e Cliente Linux

Na figura 4.18 observamos o mesmo ambiente Linux do cenário anterior, comparado com o KVM Linux, onde também observa-se a mesma tendência nos dados. Constata-se que no ambiente KVM há uma presença de *outliers* e uma menor amplitude dos dados. Conforme as análises realizadas na seção anterior, no processo de alocação, pode-se notar que existe uma forte relação da tecnologia de virtualização com a existência de um cache de dados, observando a M1 do processo de migração, tanto para o cenário 1 na figura 4.17 quanto para

o cenário 2 na figura 4.18 observamos essa mesma tendência, intensificado a hipótese de interferência direta da tecnologia nos processos de execução das operações nos ambientes.

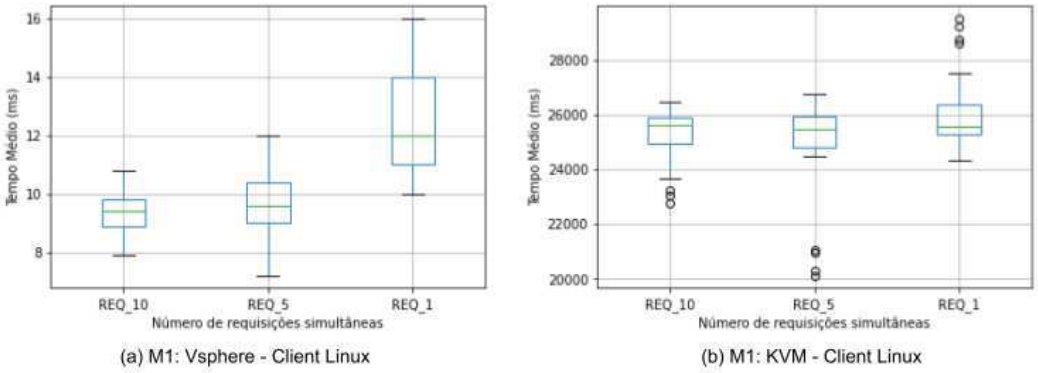


Figura 4.18: Cenário 2 - M1: vSphere e KVM - Cliente Linux

M2 - Tempo de Decisão

Para o tempo de decisão, demonstrado nas figuras 4.19 e 4.20, foram comparados os ambientes *vSphere* Linux e Windows e KVM Linux e *vSphere* Linux. Para o primeiro, pode-se observar a mesma tendência nos dados, de diminuição do tempo médio de execução à medida em que o número de requisições aumenta, observando também que a variável REQ_1 apresenta uma maior amplitude dos dados e a presença de *outliers*, assim como na métrica anterior, é possível perceber a interferência dos ambientes.

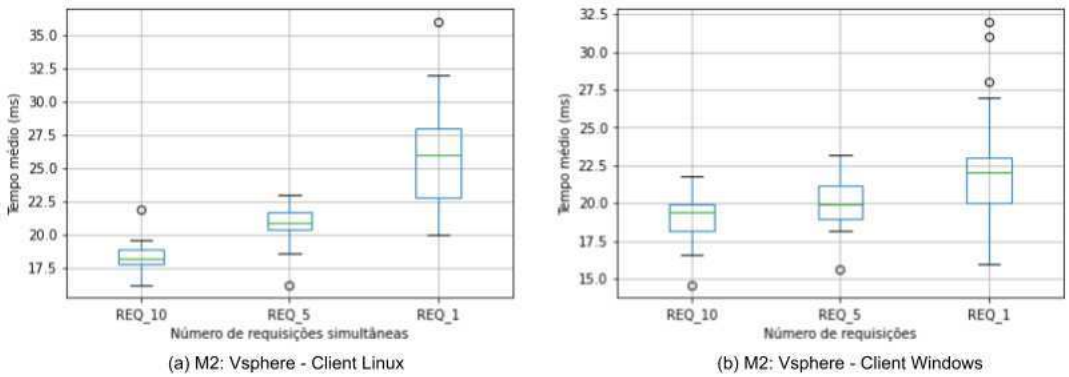
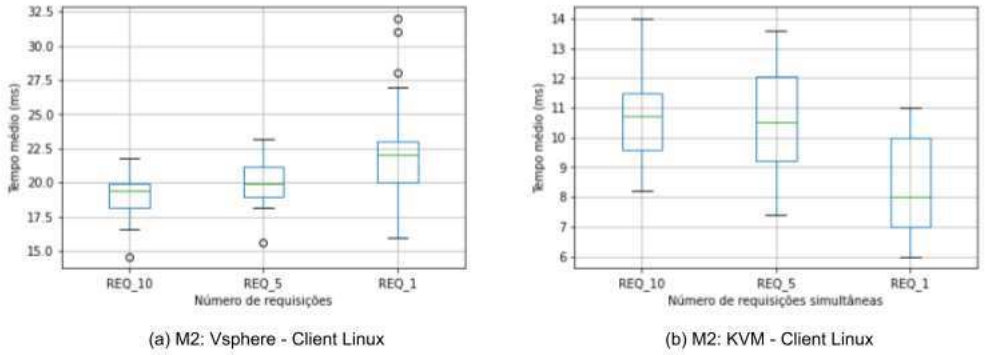


Figura 4.19: Cenário 1 - M2: vSphere - Cliente Windows e Cliente Linux

Para o KVM, na figura 4.20, podemos observar um comportamento contrário, onde a medida em que aumentamos o número de requisições o tempo médio de decisão tende a aumentar, demonstrando um comportamento anteriormente observado na M2 do processo de alocação. Nesta métrica apresenta um menor tempo médio de decisão comparado com os clientes Linux e Windows do ambiente *vSphere*, apesar de ser executado em um ambiente de uso doméstico.



(a) M2: Vsphere - Client Linux

(b) M2: KVM - Client Linux

Figura 4.20: Cenário 2 - M2: vSphere e KVM - Cliente Linux

M3 - Tempo de recuperação de hosts do Cluster

O tempo de recuperação de *hosts* compreende a diferença entre a solicitação e o retorno dos dados dos *hosts* do *Cluster*. Esta métrica é utilizada no processo de migração parar comparar os clientes do ambiente *vSphere*, e para comparar o cliente *vSphere* Linux com o KVM Linux. Assim como na métrica anterior, ao analisar a figura 4.21 percebemos uma mesma tendência no comportamento dos dois clientes, em ambos há a presença de *outliers*, marcando principalmente a variável *REQ_1* no cliente Windows, e a variável *REQ_5* no cliente Linux, entretanto, o ambiente Windows apresenta um melhor tempo médio para a recuperação dos dados do que o Linux, ocorrendo em favor da maior compatibilidade do ambiente Linux com as tecnologias do *vSphere*.

Ao comparar o ambiente *vSphere* Linux com o KVM Linux, percebe-se de maneira mais acentuada a inversão de tendência entre os dois ambientes. O KVM apresenta dados bastante distribuídos ao redor da mediana, com tendência de amplitude semelhante para as três variáveis e sem a presença marcante de *outliers*. No KVM, quanto maior o número de re-

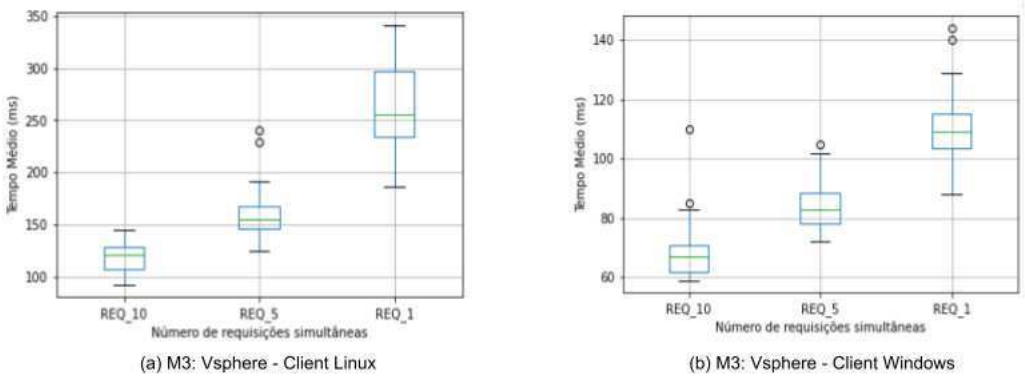


Figura 4.21: Cenário 1 - M3: vSphere - Cliente Linux e Windows

quisições simultâneas, maior o tempo médio para execução do processo, demonstrando o mesmo comportamento da M2 do processo de alocação, enquanto que o ambiente *vSphere* Linux, permanece com a diminuição do tempo médio com o aumento do número de requisições, relacionado ao fator cache da tecnologia.

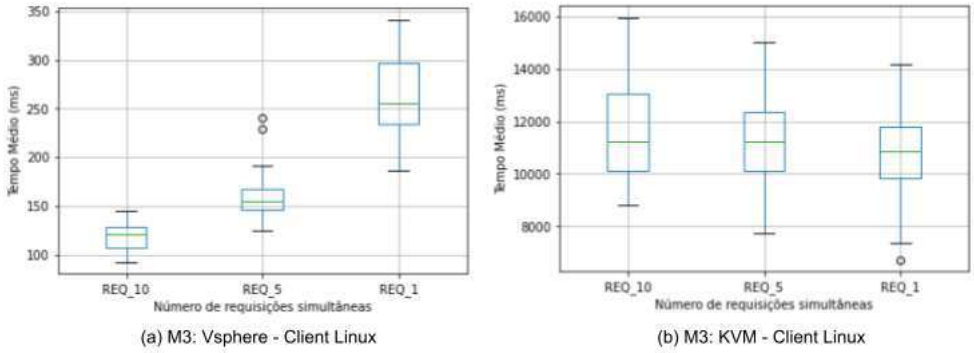


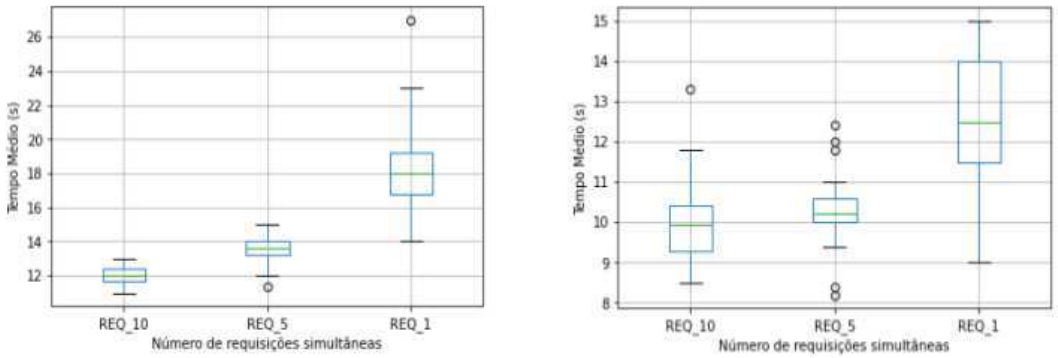
Figura 4.22: Cenário 2 - M3: vSphere e KVM - Cliente Linux

M4 - Tempo de recuperação do *storage* do Cluster

A recuperação dos dados do *storage*, assim como no processo de alocação, representa uma métrica complementar a análise do ambiente *vSphere*. Como a experimentação realizada no KVM utilizou apenas o armazenamento interno dos hosts, não foi possível realizar uma decisão separada de *storages* externos ao *host* escolhido, não ocorrendo nesta métrica a comparação entre ambientes de virtualização. Na figura 4.23, observa-se nos clientes Linux e

Windows do ambiente *vSphere* a mesma tendência nos dados, quando há um aumento no número de requisições, o tempo médio de recuperação dos dados do *storage* diminui.

No ambiente Windows há maior presença de pontos discrepantes, enquanto observamos maior limite inferior para a variável *REQ_1* e maior amplitude para os seus dados. É interessante destacar que as variáveis *REQ_10* e *REQ_5* do cliente Linux apresentam menor variabilidade e uma menor amplitude no seu conjunto de dados. Em termos absolutos, o cliente Linux tem um tempo médio de recuperação de *storage* menor que o do ambiente Windows. Isso ocorre devido as implementações das tecnologias, que podem retornar resultados diferentes à medida em que uma determinada quantidade de dados são constantemente requisitados, refletindo diretamente na sua arquitetura de desempenho.



(a) M4: Vsphere - Client Linux

(b) M4: Vsphere - Client Windows

Figura 4.23: Cenário 1 - M4: vSphere - Cliente Linux e Windows

M5 - Tempo total do processo

O tempo total do processo nos ambientes *vSphere* Linux e *vSphere* Windows mostram a mesma tendência no padrão, com o aumento do tempo médio de execução no processo total a partir do aumento progressivo do número de requisições simultâneas. Na figura 4.24, é possível observar que há uma ausência de amplitude relevante para as variável *REQ_10* e *REQ_1*. Para a variável *REQ_5*, tem-se uma maior variação interquartílica, mas há a ausência de limites interior e superior.

Com esses dados, pode-se perceber que o tempo médio para a realização da operação no ambiente Windows tem um tempo menor que a execução no cliente Linux, mas pela homo-

geneidade dos dados não é possível extrair outras características relevantes. Na observação da distribuição do tempo médio para o ambiente Linux KVM, de acordo com a figura 4.24 observamos que não há uma tendência clara no comportamento das variáveis, mas é observar que para a *REQ_1* há uma alta amplitude nos dados, com uma maior distribuição entre a mediana do conjunto.

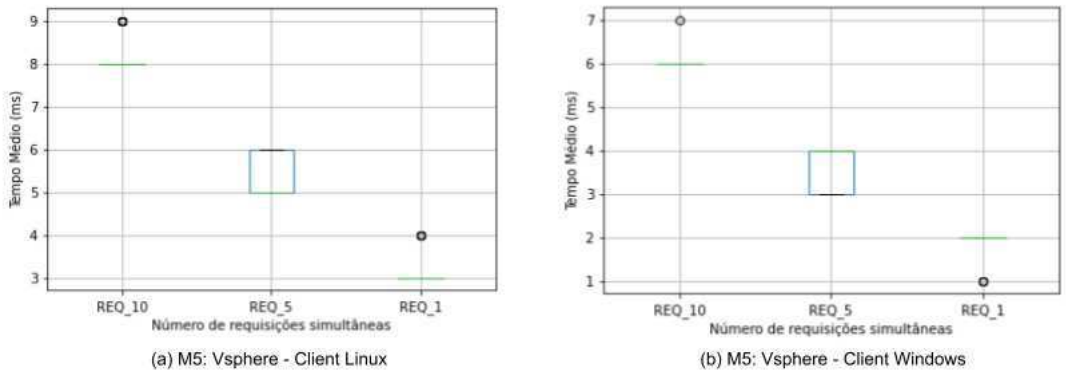


Figura 4.24: Cenário 1 - M5: vSphere - Cliente Linux e Windows

Para esta métrica, observou-se comportamentos atípicos em relação a mesma métrica do conjunto de dados do processo de alocação. Ao analisar distintamente os dados das requisições percebe-se que não há variabilidade relevante no conjunto, permanecendo com tempo médios semelhantes entre o conjunto de requisições realizadas. Na figura 4.24 do ambiente *vSphere*, é possível notar uma tendência no comportamento dos dados, mas na figura 4.25 (b) do ambiente KVM Linux, podemos notar um comportamento semelhante ao consumo de CPU observado no processo de alocação, em que o conjunto de dados atinge os valores mínimo e máximo possíveis, com alta amplitude e variabilidade nos dados. Essa característica do ambiente KVM Linux, ocorre em decorrência das tecnologias utilizadas para a comunicação, além do ambiente limitado utilizado na experimentação que influenciou diretamente no processo de execução.

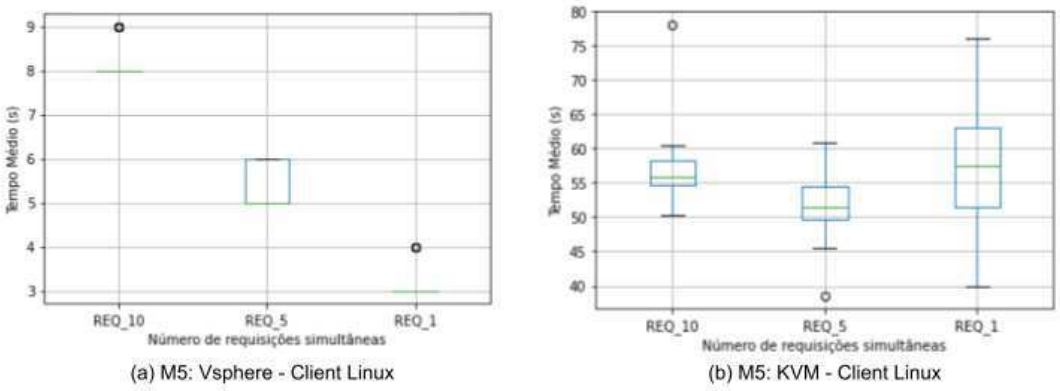


Figura 4.25: Cenário 2 - M5: vSphere e KVM - Cliente Linux

4.4.2 Consumo de Recursos

M6 - Consumo de CPU

O consumo de CPU também foi analisado como métrica no processo de migração de máquinas virtuais. A figura 4.26 exibe um comparativo entre os ambientes *vSphere* Linux e *vSphere* Windows, onde ambos apresentam a mesma tendência nos dados com maior heterogeneidade no cliente Linux, e maior amplitude na *REQ_1* e no *REQ_10*. No ambiente *vSphere* Windows, observa-se uma menor amplitude em todas as variáveis observadas, mantendo uma maior homogeneidade nos dados, com a presença de alguns pontos discrepantes para a *REQ_5*. Em termos absolutos, obtivemos um melhor ganho percentual no consumo de CPU no cliente Windows. Isso ocorreu em virtude de maior compatibilidade com sistema operacional Windows com as tecnologias do ambiente de virtualização.

A figura 4.27 mostra um comparativo entre o ambiente *vSphere* Linux e KVM Linux, para o Linux observamos um conjunto de dados mais estável com menor variabilidade e com semelhança no padrão de amplitude dos dados, além de uma distribuição ao redor da mediana semelhante ao ambiente *vSphere*. Comparando os ambientes, apesar de bastante distintos quanto à configuração de hardware, temos um desempenho percentual médio superior do KVM Linux em comparação com Linux e Windows no *vSphere*.

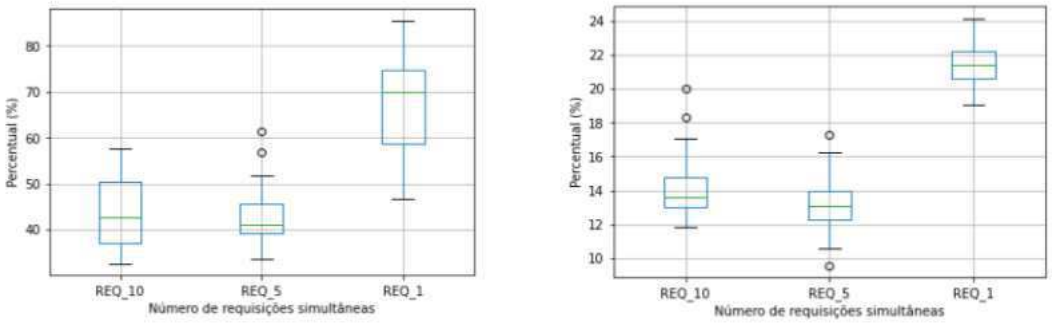


Figura 4.26: Cenário 1 - M6: vSphere - Cliente Linux e Windows

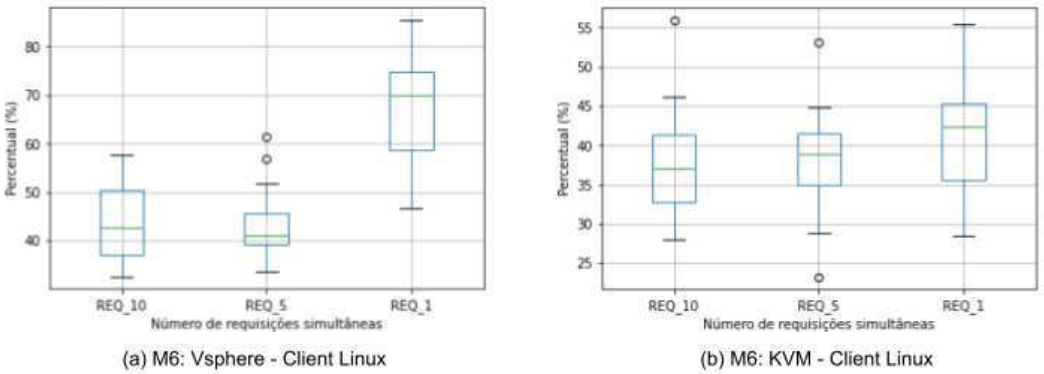


Figura 4.27: Cenário 2 - M6: vSphere e KVM - Cliente Linux

M7 - Consumo de Memória

O consumo de memória para os três ambientes analisados, *vSphere* Linux e Windows e KVM Linux, apresentados respectivamente nas figuras 4.28 e 4.29, mostram uma tendência na distribuição dos dados, tendo o ambiente *vSphere* com uma semelhança maior entre os clientes, com baixa amplitude dos dados e baixo intervalo interquartílico. Em termos absolutos, o cliente *vSphere* Linux tem um desempenho percentual médio de consumo melhor que os demais ambientes, seguido de *vSphere* Windows e KVM Linux.

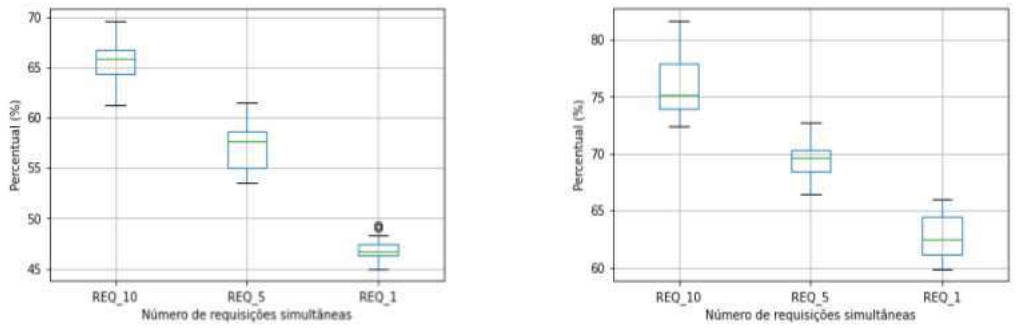


Figura 4.28: Cenário 1 - M7: vSphere - Cliente Linux e Windows

Comparando com o processo de alocação, na figura 4.28, pode-se perceber um comportamento semelhante à mesma métrica analisada no conjunto de dados do processo de alocação, onde foi possível observar que para a memória percebemos um consumo percentual proporcional ao crescimento do número de requisições simultâneas. Para o cenário da figura 4.29, também percebemos um comportamento semelhante ao mesmo cenário da métrica M7 do processo de alocação, com maior variabilidade do KVM Linux, que ocorre em decorrência das ferramentas e bibliotecas utilizadas para comunicação com o ambiente de virtualização, e em decorrência da limitação de hardware que proporciona picos nos valores do conjunto.

4.4.3 Validação

A partir da análise das métricas para o processo de migração, observa-se um comportamento bastante similar entre todos os ambientes analisados, diferentemente do processo de alocação que possuía uma maior heterogeneidade dos dados. Ao decorrer das análises anteriores,

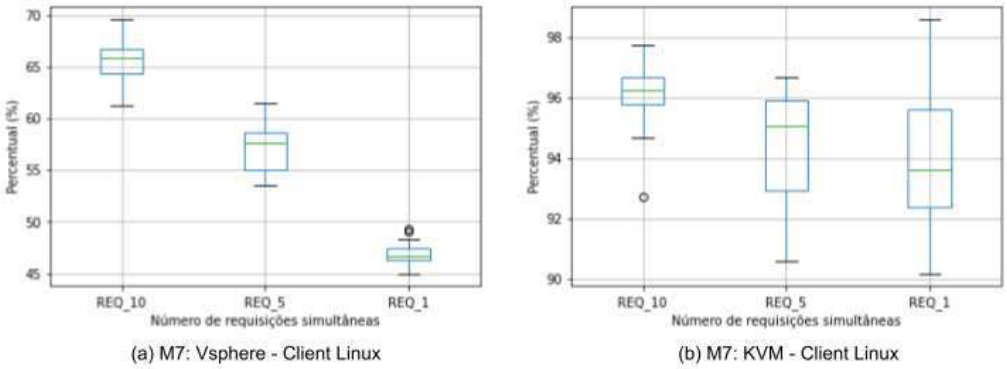


Figura 4.29: Cenário 2 - M7: vSphere e KVM - Cliente Linux

observou-se que determinadas variáveis possuíam uma relação maior entre elas. A figura 4.30 mostra a correlação entre as variáveis do processo de migração no ambiente KVM Linux.

Estabelecemos a mesma margem de comparação do processo de alocação, em que a análise inicia de forma decrescente para os percentuais de correlação de 75%, 50% e 25%. Para o ambiente KVM Linux não foi possível observar nenhuma correlação acima de 75%, mas a partir da análise com a margem de 50% foi possível identificar uma correlação forte positiva de 53% entre o aumento do número de requisições simultâneas e o aumento do consumo percentual de memória ao decorrer do processo. Para os demais pares, há mais relações fracas e positivas, do que relações fortes e negativas, diferentemente do processo de alocação.

A análise dos clientes executados no ambiente *vSphere* estão demonstrados nas figuras 4.31 e 4.32. Na primeira figura, que representa o ambiente *vSphere* Linux, podemos observar que existem fortes correlações negativas entre as variáveis, acompanhando a tendência esperada a partir do processo de alocação. Observa-se também forte tendência positiva na relação entre consumo de memória e número de requisições simultâneas, além do aumento do tempo total do processo à medida em que aumentamos o número de requisições. Também podemos observar que há uma correlação forte positiva entre o consumo de memória e o tempo total.

Ao analisar a correlações no ambiente *vSphere* Windows, na figura 4.32, podemos observar as seguintes correlações de forte intensidade:

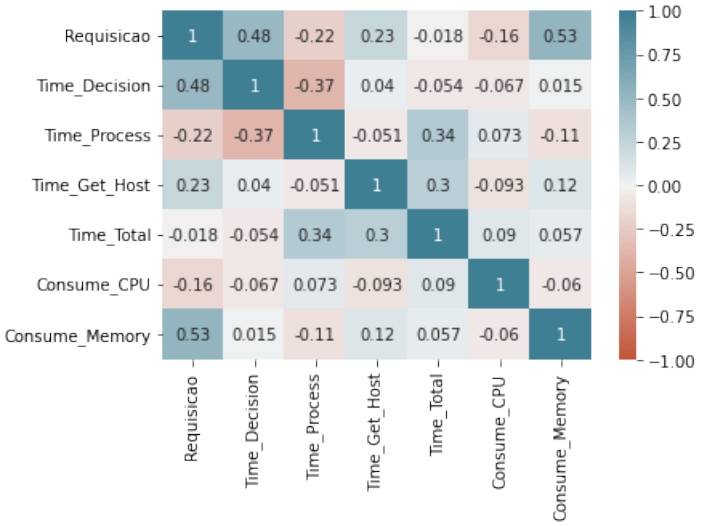


Figura 4.30: KVM Linux: Correlação entre as métricas do ambiente.

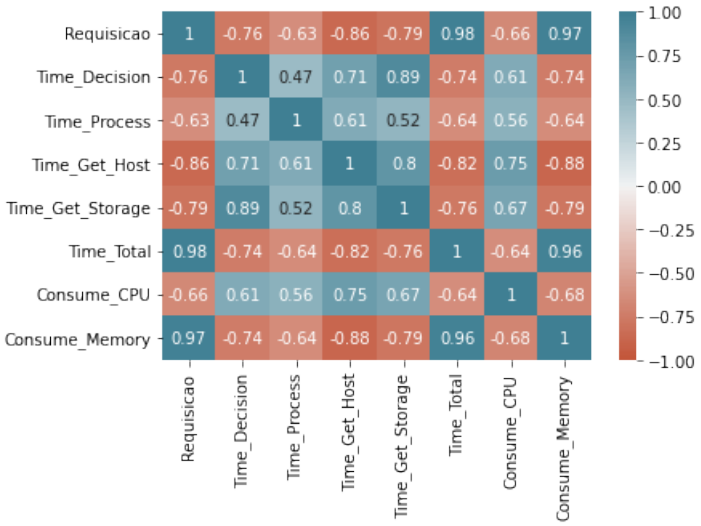


Figura 4.31: vSphere Linux: Correlação entre as métricas do ambiente.

1. Requisicao e Time_Get_Host = forte negativa,
2. Requisicao e Time_Total = forte negativa,
3. Requisicao e Consume_Memory = forte positiva,
4. Time_Get_Host e Time_Total = forte positiva,
5. Time_Get_Host e Consume_Memory = forte negativa,

6. Time_Total e Consume_Memory = forte negativa.

Com esses dados, pode-se perceber que há fortes correlações positivas entre as variáveis, mas a maioria representa uma correlação forte negativa, principalmente entre os pares 1,2, 5 e 6, enquanto que os pares 3 e 4 apresentam uma correlação forte e positiva. Esses dados permitem concluir que há uma relação de aumento do número de requisições com o consumo de memória e com o tempo total de processamento, no processo de migração no *vSphere* Linux. A correlação negativa nos demais pares, pode indicar o cache de determinados dados ao decorrer das operações.

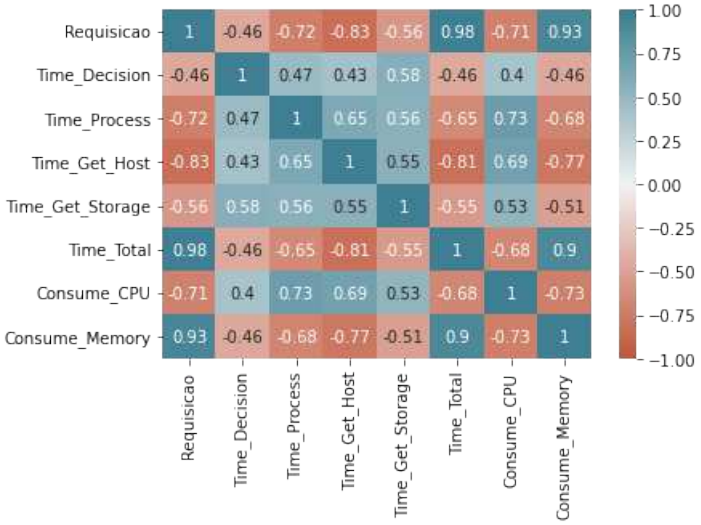


Figura 4.32: vSphere Windows: Correlação entre as métricas do ambiente.

A variabilidade presente nas métricas analisadas no ambiente *vSphere* permite evidenciar a influência do ambiente dos valores e percentuais observados para os tempos de execução e os percentuais de consumo de recursos, constatando um melhor desempenho do ambiente *vSphere* Windows em virtude da maior compatibilidade do sistema operacional com as ferramentas da tecnologia de virtualização do ambiente *vSphere*. Além disso, foi possível observar uma forte correlação negativa entre algumas variáveis, percebendo a manutenção de padrões observados no processo de alocação, com o aumento de tempos e percentuais de algumas métricas à medida em que o número de requisições simultâneas sofrem um aumento progressivo. Para as métricas observadas no ambiente KVM Linux, foi possível observar forte relação da restrição do ambiente nos valores das métricas, com algumas delas

atingindo os máximos e mínimos possível dentro do intervalo esperado.

Capítulo 5

Conclusão

Após os resultados da validação deste trabalho, com o processo de alocação e migração automática de máquina virtuais com base em critérios de confiabilidade a análise considerou apenas dois ambientes de virtualização: o *VSphere ESXi* e o *KVM*. O ambiente do *vSphere* foi um *cluster* com servidores de alto desempenho utilizado em ambiente real e o ambiente *KVM* foi virtualizado em uma máquina pessoal. Das setes métricas especificadas na tabela 4.4, todas foram aplicáveis ao *vSphere* e apenas a métrica M4 não foi aplicável ao KVM, devido a particularidade do ambiente em utilizar apenas uma unidade de armazenamento.

Para a avaliação foram utilizadas cinco métricas de tempo de execução que avaliaram aspectos distintos dos processos propostos e duas métricas de consumo de recursos que foram avaliadas em ambos os ambientes. No processo de alocação tivemos uma maior variabilidade no conjunto de dados, destacando uma maior amplitude e variação interquartílica. Mas ao decorrer da análise percebemos que o ambiente *vSphere Windows* teve um maior desempenho quando comparado aos tempos médios e consumos percentuais dos demais ambientes. Essa característica está relacionada a maior compatibilidade do cliente *Windows* com as tecnologias de virtualização do ambiente *vSphere*.

Durante a análise do processo de migração, apesar da limitação de hardware do KVM foi possível observar um uso menor de tempo para realização de algumas operações como o processo decisório para escolha do melhor *host*. Em relação as outras métricas tivemos valores bem próximos entre os ambientes, mas os ambientes *KVM Linux* e *vSphere Windows* apresentam os melhores resultados. Apesar da utilização de variadas métricas, onde mais da metade eram métricas de tempo, os resultados foram influenciados pela limitação

de hardware existente durante o desenvolvimento do trabalho. Um dos ambientes era um ambiente externo com limitação de acesso, mas com boa qualidade física para rodar os experimentos e o outro ambiente executou em um notebook de uso pessoal, que acabou limitando alguns parâmetros de capacidade como número de CPUs, memória e disco, interferindo diretamente na variabilidade do conjunto de dados.

A utilização de um *cluster* em operação também dificultou o desenvolvimento do trabalho, uma vez que houve trocas de *build* dos hipervisores sem documentação das mudanças realizadas, ocasionando mais tempo para realizar a investigação da mudança e a correção de erros de comunicação na implementação da ferramenta. No ambiente KVM a limitação de hardware também dificultou a execução dos experimentos, sendo necessário reduzir ao máximo os recursos utilizados por cada máquina alocada e migrada para poder atender aos critérios e parâmetros estabelecidos na proposta.

No aspecto da integridade de sistemas, a proposta foi avaliada considerando os cenários estabelecidos na tabela 4.2. Para execução dos experimentos foram consideradas uma rodada de 35 execução, 30 das quais considerando o TPM dos nós validados e 5 execuções considerando o TPM do nó com falha na validação. Além da validação do TPM também foi possível realizar a validação de integridade dos arquivos de medições presentes nos sistemas operacionais utilizados nos experimentos, para isso foram utilizando o arquivo do IMA para sistemas operacionais baseado em Linux e o arquivo WBCL para sistemas Windows.

Além da validação de integridade dos arquivos de log de medição, também foi realizada a validação das operações de usuário a partir da criação de uma *whitelist* presente em cada nó. Essa lista é responsável por armazenar todos os comandos que são permitidos de serem executados dentro da máquina. Dessa forma, o processo de validação da integridade no experimento foi executando considerandos os chips TPM físico e virtual, os arquivos de medição do IMA e do WBCL e uma *whitelist* customizada para os comandos permitidos dentro de cada sistema operacional.

Assim, a solução proposta se mostrou viável para desenvolvimento em *clusters* com tecnologias distintas e apesar da utilização de interfaces gerais de comunicação, ainda é necessário realizar adaptações específicas para cada tecnologia a ser utilizada. Além disso, a ferramenta possibilitou o desenvolvimento mais rápido de atividades alocação e migração dentro do *cluster*, além de garantir a confiabilidade do ambiente através da migração auto-

mática de máquinas virtuais nas quais o seu hospedeiro perdeu a garantia de confiabilidade, possibilitando o gerenciamento automático de mudanças críticas do *cluster*.

5.1 Trabalhos futuros

Após os resultados obtidos na validação deste trabalho com a análise de ambientes de virtualização e o auxílio na tomada de decisão, nos limitamos à utilização de métricas de tempo e consumo de recursos de CPU e memória, duas tecnologias de virtualização e apenas uma tecnologia de integridade de dados (TPM) utilizada no processo de validação. Houve também limitação de recursos de hardware em decorrência dos ambientes em que foram realizados os experimentos. Para o gerenciamento dos ambientes de virtualização, também nos limitamos a implementação de processos de alocação e migração, nesse aspecto podem ser desenvolvidos outros mecanismos de gerenciamento, como remoção de máquinas virtuais e indisponibilidade de *hosts* com a confiabilidade comprometida. Logo, destacamos alguns possíveis trabalhos futuros para continuação desta pesquisa:

- Adicionar mais métricas de desempenho, como taxa de transferência, eficiência de memória e disponibilidade;
- Considerar o uso de outros métodos de decisão, como Redes Neurais Artificiais (ANN) ou Programação Linear;
- Considerar outras tecnologias de virtualização, como Microsoft Hyper-V e Citrix XenServer;
- Considerar outras tecnologias de garantia de integridade de dados, como Intel SGX;
- Adicionar outros processos de gerenciamento automático, como remoção de máquinas virtuais;
- Adicionar outros processos para gerenciamento dos *hosts* comprometidos, como remoção ou torná-los indisponíveis no *cluster*.

Bibliografia

[1] K. Ahmadi and V. H. Allan. Trust-based decision making in a self-adaptive agent organization. *ACM Trans. Auton. Adapt. Syst.*, 11(2), jun 2016.

[2] Y. Ahmed, S. Naqvi, and M. Josephs. Aggregation of security metrics for decision making: A reference architecture. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ECSA '18, New York, NY, USA, 2018. Association for Computing Machinery.

[3] J. R. Almeida, J. B. Camargo, and P. S. Cugnasca. Entre a pandemia e o pandemônio: Uma reflexão no campo da educação. *Educamazônia - Educação, Sociedade e Meio Ambiente*, 25(2):291–311, 2020.

[4] A. S. Banks, M. Kisiel, and P. Korsholm. Remote attestation: A literature review, 2021.

[5] L. M. C. Calejon and A. S. Brito. Safety and security in critical applications and in information systems - a comparative study. *IEEE Latin America Transactions*, 11(4):1127–1133, 2013.

[6] A. Carissimi. Virtualização: da teoria a soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC*, 2008:173–207, 2008.

[7] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. Principles of remote attestation. *Int. J. Inf. Sec.*, 10:63–81, 06 2011.

[8] T. S. Community. *Remote Attestation*. https://tpm2-software.github.io/tpm2-tss/getting-started/2019/12/18/Remote-Attestation.html, 2019.

[9] C. C. Consortium. *A Technical Analysis of Confidential Computing*. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3$_u nlocked.pdf, 2023.$

[10] M. De Benedictis and A. Lioy. Integrity verification of docker containers for a lightweight cloud environment. *Future Generation Computer Systems*, 97:236–246, 2019.

[11] A. De Mauro, M. Greco, and M. Grimaldi. What is big data? A consensual definition and a review of key research topics. *AIP Conference Proceedings*, 1644(1):97–104, 02 2015.

[12] G. de Teleinformática e Automação. *O que é virtualização?* https://www.gta.ufrj.br/ensino/eel879/trabalhos$_v 1_2 017_2 /kvm/, 2022.$

[13] M. Emami-Taba. Decision-making in self-protecting software systems: A game-theoretic approach. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, page 77–79. IEEE Press, 2017.

[14] K. Ezirim, W. Khoo, G. Koumantaris, R. Law, and I. Perera. Trusted platform module – a survey. 11 2012.

[15] A. Ferraris, A. Mazzoleni, A. Devalle, and J. Couturier. Big data analytics capabilities and knowledge management: impact on firm performance. *Management Decision*, 57(8):1923–1936, 2019.

[16] C. N. C. Foundation. *Keylime Documentation*. https://keylime.readthedocs.io/en/latest/, 2022.

[17] C. N. C. Foundation. *SPIRE Concepts*. https://spiffe.io/docs/latest/spire-about/spire-concepts/, 2023.

[18] R. Hat. *O que é virtualização?* https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization, 2018.

[19] R. Hat. *O que é KVM?* https://www.redhat.com/pt-br/topics/virtualization/what-is-KVM, 2022.

[20] S. C. Helble, I. D. Kretz, P. A. Loscocco, J. D. Ramsdell, P. D. Rowe, and P. Alexander. Flexible mechanisms for remote attestation. *ACM Trans. Priv. Secur.*, 24(4), sep 2021.

[21] S. Hiremath and S. Kunte. A novel data auditing approach to achieve data privacy and data integrity in cloud computing. In *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 306–310, 2017.

[22] W. Hongyuan. An external data integrity tracking and verification system for universal stream computing system framework. In *2019 21st International Conference on Advanced Communication Technology (ICACT)*, pages 32–37, 2019.

[23] J. A. M. S. Junior. Escalonamento em sistemas de tempo real multiprocessados com baixo custo de implementação. Dissertação (mestrado), Programa de Pós-graduação em Mecatrônica, Salvador, 2017.

[24] M. A. P. Laureano and C. A. Maziero. Virtualização: Conceitos e aplicações em segurança. *Sociedade Brasileira de Computação*, 2008.

[25] W. Luo, Q. Shen, Y. Xia, and Z. Wu. Container-ima: A privacy-preserving integrity measurement architecture for containers. In *RAID*, pages 487–500, 2019.

[26] P. Mell and T. Grance. Draft nist working definition of cloud computing. *National Institute of Standards and Technology*, 15:1–7, 2009.

[27] D. Menezes and F. Mattos. Virtualização: Vmware e xen. 01 2008.

[28] G. Reinehr. Arcabouço de segurança para medidores inteligentes no contexto de computação em nuvem. Dissertação (mestrado), Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, 2017.

[29] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, 2005.

[30] V. Scarlata, C. Rozas, M. Wiseman, D. Grawrock, and C. Vishik. *TPM Virtualization: Building a General Framework*, pages 43–56. Vieweg+Teubner, Wiesbaden, 2008.

[31] H. Sidhpurwala. How to use the linux kernelś integrity measurement architecture. *BLOG DA RED HAT*, 2020.

[32] L. A. Siqueira. *Máquinas virtuais com VirtualBox*. Linux New Media do Brasil E, 2011.

[33] S. Sisinni, D. Margaria, I. Pedone, A. Lioy, and A. Vesco. Integrity verification of distributed nodes in critical infrastructures. *Sensors*, 22(18), 2022.

[34] R. Sohr. Ataque al corazon de Washington. *Mensaje*, 70(696):22, 02 2021.

[35] H. Sun and R. He. Certification and remote attestation methods of the etpm trusted cloud. In *Proceedings of the 8th International Conference on Communication and Network Security*, ICCNS 2018, page 42–49, New York, NY, USA, 2018. Association for Computing Machinery.

[36] VMware. *VMware ESX e VMware ESXi: Os hypervisores líderes do mercado com produção comprovada*. https://www.vmware.com/files/br/pdf/products/VMW$_0$9Q1$_B$$RO_E$$SX_E$$SXi_B$$R_A4_P6_R$2.$pdf$, 2009.

[37] S. Zaman and D. Grosu. Combinatorial auction-based allocation of virtual machine instances in clouds. *Journal of Parallel and Distributed Computing*, 73(4):495–508, 2013.