MARCIO FELICIANO DO PRADO

KWH-PRO:
UM ARCABOUÇO PARA ALOCAÇÃO DINÂMICA DE VMS E REDUÇÃO DO CONSUMO DE ENERGIA

LAVRAS – MG

2019

MARCIO FELICIANO DO PRADO

KWH-PRO:
UM ARCABOUÇO PARA ALOCAÇÃO DINÂMICA DE VMS E REDUÇÃO DO CONSUMO DE ENERGIA

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Redes de Computadores e Sistemas Embarcados, para a obtenção do título de Mestre.

Prof. PhD. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo próprio autor.

Prado, Marcio Feliciano do.
kWh-PRO : um arcabouço para alocação dinâmica de VMs e redução do consumo de energia / Marcio Feliciano do Prado. – 2019.
101 p. : il.

Orientador: Prof. PhD. Luiz Henrique Andrade Correia.

Dissertação (mestrado acadêmico)–Universidade Federal de Lavras, 2019.
Bibliografia.

1. *data centers*. 2. migração de VMs. 3. energia. I. Correia, Luiz Henrique Andrade. II. Título.

MARCIO FELICIANO DO PRADO

KWH-PRO: UM ARCABOUÇO PARA ALOCAÇÃO DINÂMICA DE VMS E REDUÇÃO DO CONSUMO DE ENERGIA

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Redes de Computadores e Sistemas Embarcados, para a obtenção do título de Mestre.

APROVADA em 20 de Fevereiro de 2019.

Prof. Dr. Neumar Costa Malheiros	UFLA
Prof. PhD. Dorgival Olavo Guedes Neto	UFMG

Prof. PhD. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG
2019

Dedico este trabalho à minha esposa Flávia, aos meus pais Maria das Graças e João Domingos e à minha irmã Rosângela, pois são meus pilares.

AGRADECIMENTOS

Primeiramente a Deus, que é o criador de todas as coisas e dono de toda ciência. Ao Professor PhD. Luiz Henrique Andrade Correia, que me orientou com sabedoria, pontualidade e dedicação. Aos professores Dr. Neumar Costa Malheiros e PhD. Dorgival Olavo Guedes Neto (UFMG), que contribuíram com o trabalho durante o exame de qualificação e durante a defesa. Estendo também estes agradecimentos aos demais professores e técnicos administrativos do Departamento de Ciência da Computação da UFLA, que direta ou indiretamente contribuíram com o trabalho. Ao meu filho de coração, João Paulo, a minha madrinha Denilza, a meu sobrinho João Antônio, a meu cunhado Joaquim e aos meus sogros Odete e Antônio pelas orações. Aos colegas de curso, que sempre compartilhavam ideias. Ao IFSULDEMINAS, que apoiou o trabalho. Por fim, porém não menos importante, a cada cidadão brasileiro, que através do pagamento de impostos, possibilita às universidades federais ofertarem cursos de qualidade e de forma gratuita.

Se vi mais longe, foi por estar sobre ombros de gigantes.
Isaac Newton

RESUMO

Atualmente, há uma crescente necessidade de armazenamento e processamento de dados. Isso vem fazendo com que cresça a implantação de *data centers* em todo o mundo. Estima-se que esses centros representem cerca de 2% do consumo mundial de energia elétrica, e a tendência é que esse consumo continue aumentando. Assim, é evidente a necessidade de pesquisas nessa área. Este trabalho apresenta um arcabouço, que tem como meta principal a redução do consumo de energia em *data centers* com VMs, e que a economia interfira minimamente no desempenho desses centros. Para atingir o objetivo, o arcabouço realiza, de forma automática, as seguintes tarefas: ***(i)*** monitoramento do consumo energético e da carga de trabalho de servidores, ***(ii)*** classificação de servidores utilizando os dados do monitoramento, ***(iii)*** alocação dinâmica de VMs com migração a quente e **(*iv*)** ligamento e desligamento de servidores. Para validação da eficiência do arcabouço, utilizou-se uma nuvem computacional implementada com o OpenStack. Foram executados vários experimentos, nos quais foi possível observar economia de energia de até 38%, ao custo de uma ligeira degradação do desempenho da nuvem, que aparentemente é imperceptível.

Palavras-chave: *data centers*, OpenStack, monitoramento, migração de VMs, energia.

ABSTRACT

Currently, there is a growing need for data storage and processing. This has been driving the roll-out of data centers around the world. These centers are estimated to account for about 2% of the world's electricity consumption, and the trend is that consumption will continue to increase. Thus, the need for research in this area is evident. This work presents a framework that has as main goal the reduction of energy consumption in data centers with VMs and that the economy interferes minimally in the performance of these centers. In order to achieve the objective, the framework automatically performs the following tasks: ***(i)*** monitoring of the energy consumption and workload of servers, ***(ii)*** classification of servers using the monitoring data, ***(iii)*** VMs with hot migration and ***(iv)*** server bindings and shutdowns. To validate the scaffold efficiency, a computational cloud implemented with OpenStack was used. Several experiments were performed, in which energy savings of up to 38% could be observed, at the cost of a slight degradation of cloud performance that seemingly imperceptible.

Keywords: data centers, OpenStack, monitoring, VM migration, power.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

A grande demanda por armazenamento e processamento de dados tem aumentado a quantidade de *data centers* em todo o mundo (NEJAD et al., 2015). Dessa forma, os custos associados à operação e manutenção desses centros também tendem a crescer. Atualmente, os *data centers* consomem cerca de 2% de toda energia elétrica mundial e a tendência é que este consumo continue aumentando (ARROBA et al., 2015).

A Google demonstrou preocupações com esse consumo de energia dos *data centers*. Prova disso é que, em meados de 2013, ela finalizou a implantação de um *data center* em Hamina, na Finlândia (GOOGLE, 2017a). Essa cidade foi escolhida por ter um clima frio e ser próxima do mar gelado do Golfo da Finlândia. Esse *data center*, em Hamina, é refrigerado com a água gelada do mar. Desse modo, economiza-se energia elétrica, pois se dispensam os convencionais sistemas de ar condicionado (GOOGLE, 2017b). A empresa informou ter economizado mais de um bilhão de dólares, até os dias atuais, a partir dessas medidas (BARROSO; CLIDARAS; HöLZLE, 2013).

Além disso, monitorar e analisar como é gasta toda essa energia, é importante para otimizar esse consumo (ROSSIGNEUX et al., 2014), (MASTELIC et al., 2014), (KAVANAGH; ARMSTRONG; DJEMAME, 2016), (BARROSO; CLIDARAS; HöLZLE, 2013). A refrigeração é responsável por cerca de 50% de toda energia consumida nos *data centers*, ao passo que servidores e dispositivos de armazenamento consomem cerca de 26%. O consumo de energia dos servidores depende de vários fatores, como carga de trabalho e tipos de aplicações. Para completar os 100%, estão os gastos com iluminação e conversão de energia, que são responsáveis pelo consumo de aproximadamente 14%, e os *switches* e roteadores que são responsáveis por 10% (DAYARATHNA; WEN; FAN, 2016).

De maneira a reduzir a quantidade de servidores físicos e, consequentemente, o consumo de energia, a proposta mais viável tem sido a virtualização de servidores. Uma das plataformas mais utilizadas nos *data centes* para promover esta virtualização é o OpenStack. Ele foi criado em 2011 pela NASA e Rackspace, e hoje é apoiado por mais de 850 organizações (VARASTEH; GOUDARZI, 2015).

Apesar da virtualização de servidores promover considerável economia de energia, ainda há vários estudos que buscam otimizar este consumo. Grande parte desses concentra-se no desenvolvimento de algoritmos que possibilitem a migração de VMs (*Virtual Machines*) e o desligamento de servidores subutilizados (ALI et al., 2015). Outros trabalhos buscam contri-

buir através do monitoramento, já que é difícil otimizar, sem entender como é a distribuição do consumo de energia (ROSSIGNEUX et al., 2014), (MASTELIC et al., 2014), (KAVANAGH; ARMSTRONG; DJEMAME, 2016), (BARROSO; CLIDARAS; HöLZLE, 2013).

1.1 Definição do Problema

O principal motivo da concentração de pesquisas no desenvolvimento de algoritmos eficientes para alocação dinâmica de VMs e desligamento de servidores ociosos é devido ao consumo energético de servidores. Mesmo ociosos, eles consomem cerca de 80% da potência total da fonte de alimentação. Assim, um servidor com carga de trabalho de apenas 15% da sua capacidade de processamento pode consumir 80% da energia que consumiria se estivesse trabalhando com carga de 100% da sua capacidade. Além disso, é comum existirem servidores subutilizados em *data centers*. Geralmente, esses servidores podem estar trabalhando na faixa de 10 a 15% da sua capacidade total de processamento, o que significa um desperdício significativo de energia.

Apesar dessa concentração de estudos na alocação dinâmica de VMs, não se conhece um arcabouço amplamente aceito, que foi validado em ambiente real, que interfira minimamente no desempenho da nuvem computacional e que propicie: **(*i*)** monitoramento centralizado do consumo energético e carga de trabalho de servidores, **(*ii*)** alocação dinâmica de VMs e **(*iii*)** desligamento de servidores ociosos. Além disso, a extensa maioria dos trabalhos avaliam os algoritmos em simuladores. Simuladores são ferramentas reconhecidas por conseguirem disseminar ideias elementares e facilidade de realização testes. Porém, por melhor que seja um simulador, é difícil abranger todas as variáveis de um ambiente real.

1.2 Motivação

O crescente consumo de energia elétrica pelos *data centers*, e o aumento dos custos associados à operação e manutenção desses centros evidenciam a necessidade de pesquisas nesta área. Assim, o desenvolvimento de um arcabouço que possibilite o monitoramento centralizado do consumo energético e carga de trabalho de servidores, alocação dinâmica de VMs, desligamento de servidores ociosos e que interfira minimamente no desempenho da nuvem computacional abre caminhos no que diz respeito à otimização do consumo de energia em *data centers* e à realização de novas pesquisas.

1.3 Objetivos Gerais

O objetivo deste trabalho é propor um arcabouço que possibilite economia de energia em *data centers* com VMs e que interfira minimamente no desempenho desses centros. Além disso, outro objetivo é a validação do arcabouço em ambiente real.

1.4 Objetivos Específicos

Além dos objetivos gerais, o trabalho apresenta os seguintes objetivos específicos:

- Propor os seguintes *middleware*:
  - kWh-MONITOR (Monitoramento do consumo energético de servidores);
  - HW-SNMP (Monitoramento da utilização do *hardware* de servidores);
  - MAPPER-VMs (Mapeamento de localização de VMs).
- Propor métodos para:
  - Classificação de servidores com base na utilização do *hardware*;
  - Tomada de decisão de migração de VMs e desligamento de servidores ociosos.
- Quantificar a economia de energia alcançada por meio do arcabouço.
- Analisar o impacto do arcabouço no desempenho da nuvem computacional.

1.5 Estrutura do Trabalho

O trabalho está dividido em sete Seções. A Seção 2 apresenta o referencial teórico, que visa auxiliar no entendimento do trabalho. Na Seção 3, apresentam-se alguns trabalhos relacionados. O arcabouço é apresentado na Seção 4. Na Seção 5, são apresentados os materiais e métodos. A Seção 6 apresenta os resultados e discussões. Por fim, na Seção 7, apresentam-se as conclusões e trabalhos futuros.

1.6 Contribuições

A principal contribuição do trabalho, é a considerável economia de energia que o arcabouço pode propiciar, em *data centers* com VMs, ao custo de uma ligeira degradação do desempenho. Outra contribuição importante é o desenvolvimento do método de classificação de servidores com base na utilização do *hardware* e do método de decisão de migração de VMs e desligamento de servidores ociosos.

2 REFERENCIAL TEÓRICO

Esta seção busca auxiliar no entendimento do trabalho. Dessa forma, são abordados vários assuntos distintos. Primeiramente, faz-se uma introdução, na qual é apresentado o problema do consumo energético dos *data centers*. Além disso, abordam-se alguns temas atuais relacionados a esses centros, como: ***(i)*** virtualização de servidores, ***(ii)*** alocação de VMs (*Virtual Machines*), ***(iii)*** conteinerização de *software*, ***(iv)*** computação na nuvem, ***(v)*** OpenStack, ***(vi)*** orquestração na computação em nuvem, ***(vii)*** alocação dinâmica de VMs com migração a quente, ***(viii)*** migração de contêineres, ***(ix)*** relação entre a economia de energia e o SLA (*Service Level Agreement*) e ***(x)*** projeto GEYSER (*Green nEtworked data centres as energY proSumers in smaRt city environments*).

Além dos assuntos relacionados aos *data centers*, são abordados alguns temas que têm relação mais direta com monitoramento, sendo eles: ***(i)*** protocolo SNMP (*Simple Network Management Protocol*) e as ferramentas *Collectd* e *Sensu*, ***(ii)*** wattímetros, ***(iii)*** IPMI (*Intelligent Platform Management Interface*), ***(iv)*** Sonoff Pow, ***(v)*** modúlo de telemetria do OpenStack e ***(vi)*** métricas de avaliação de desempenho de redes.

2.1 Introdução

Em 2010, a geração de dados a cada dois dias ultrapassou o total de dados gerados em 2003. Por sua vez, um estudo feito pela IDC (*International Data Corporation*) mostrou que o volume de dados gerados em 2020 será cinquenta vezes maior que o gerado em 2011. Este é o motivo do aumento da implementação de *data centers* em todo o mundo (NEJAD et al., 2015).

Os custos ligados à manutenção e operação desses centros também estão crescendo. Nos dias de hoje, os *data centers* consomem aproximadamente 2% de toda energia elétrica mundial e a tendência é que esse consumo aumente. Com isso, os *data centers* estão se tornando insustentáveis em termos de consumo energético (ARROBA et al., 2015). Além do valor econômico, outro fator que deve ser considerado é o aumento do nível de dióxido de carbono na atmosfera. O grande consumo de energia elétrica dos *data centers* tem relação direta com o aumento desse gás (WADHWA; VERMA, 2014).

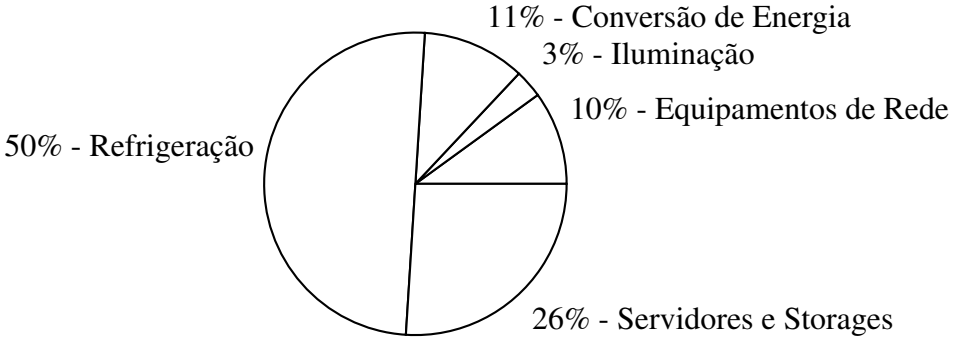
Empresas como Google, Facebook e Microsoft já demonstraram preocupações com esse consumo e já estão realizando ações para otimizar esse gasto. O Facebook, em 2014, anunciou um *data center* na cidade de Luleå, na Suécia. Luleå foi escolhida porque possui temperaturas muito baixas. Além disso, a infraestrutura elétrica da Suécia é uma das mais confiáveis, e

conta com várias fontes renováveis. Isso foi outra vantagem, tendo em vista que a empresa tem comprado vários parques eólicos nos Estados Unidos para suprir seus *data centers* (MORAES, 2014), (SPITZCOVSKY, 2011).

Em 2015, a Microsoft implantou um *data center* no fundo do Oceano Pacífico por três meses. O principal motivo do experimento foi a busca por economia de energia com a refrigeração. Este experimento faz parte do projeto Natick, que é um projeto de pesquisa que busca criar *data centers* mais rápidos e sustentáveis (MICROSOFT, 2014).

Além dessas ações, monitorar e analisar como é gasta essa energia são tarefas cruciais para otimizar esse consumo (ROSSIGNEUX et al., 2014), (MASTELIC et al., 2014), (KAVANAGH; ARMSTRONG; DJEMAME, 2016), (BARROSO; CLIDARAS; HöLZLE, 2013). A energia consumida por um *data center* pode ser dividida em duas grandes partes. A primeira é o sistema de refrigeração, que consome cerca de 50% do total. A segunda são os equipamentos de tecnologia da informação que consomem cerca de 36%. Destes 36%, os *switches* e roteadores são responsáveis por 10%, servidores e dispositivos de armazenamento são responsáveis pelos outros 26%. Assim, servidores e dispositivos de armazenamento estão no segundo lugar na hierarquia do consumo. O consumo de energia dos servidores depende de múltiplos fatores, como: especificações de *hardware*, carga de trabalho e tipos de aplicações. Para completar os 100% estão os gastos com iluminação e conversão de energia, que são responsáveis pelo consumo de aproximadamente 14%. A Figura 1 apresenta os consumidores de energia em um *data center* (DAYARATHNA; WEN; FAN, 2016).

Figura 1 – Consumo de Energia nos *Data Centers*



Fonte: adaptada de (DAYARATHNA; WEN; FAN, 2016)

Historicamente, uma prática comum entre os administradores de *data centers*, é a de hospedar uma aplicação por servidor. Essa medida visa garantir maior segurança às aplicações, já que a vulnerabilidade de uma aplicação tende a afetar somente o servidor em que está hospe-

dada. No entanto, com o passar do tempo e aumento do poder de processamento dos servidores, o *hardware* começou a ficar subutilizado (VERAS; KASSICK, 2011).

Além da subutilização dos recursos e consumo energético, começou a haver problemas de espaço físico nos *data centers*. Dessa forma, a solução proposta mais viável tem sido a consolidação de servidores. Esta, por sua vez, consiste em executar diversos sistemas operacionais, simultaneamente, em um servidor físico. Para tornar a consolidação de servidores possível, utiliza-se a técnica denominada virtualização de servidores. Assim, os recursos (*hardware*) de um servidor físico, são divididos entre os servidores virtuais ou mais comumente chamados de VMs. Dessa forma, garante-se o isolamento das aplicações, o aumento da taxa de utilização do *hardware* e redução dos custos operacionais com espaço físico e consumo energético (VALLEE et al., 2008).

2.2 Virtualização de Servidores

A virtualização é uma técnica antiga que surgiu entre o fim da década de 60 e o início de 70. Foi naquela época que surgiram os primeiros *mainframes*, que por serem computadores grandes e caros, geralmente tinham que ser compartilhados por vários usuários. Nesse sentido, houve um forte incentivo ao uso da virtualização, já que os usuários precisavam utilizar sistemas computacionais e ficava inviável ter um *mainframe* para cada pessoa (VARASTEH; GOUDARZI, 2015).

Na década de 80, o *hardware* passou a ser mais barato e os *mainframes* foram substituídos por *desktops*, diminuindo o interesse pelo uso da virtualização, pois cada usuário podia ter o seu computador físico exclusivo. Recentemente, o uso virtualização voltou a se tornar popular. Os grandes e caros *mainframes* do passado são hoje os também potentes e caros servidores (VALLEE et al., 2008).

Para possibilitar a execução de diversos sistemas operacionais, simultaneamente, em um único servidor físico é utilizado o *hypervisor*. O *hypervisor*, ou VMM (*Virtual Machine Monitor*), é uma camada de *software* localizada entre o *hardware* e as VMs. (AWASTHI; GUPTA, 2016). A Figura 2 mostra, de forma abstrata, a arquitetura de virtualização de servidores, na qual é possível observar o servidor físico, o *hypervisor* e várias VMs.

Atualmente, existem três tipos de virtualização baseadas em *hypervisors*. São elas: virtualização completa, paravirtualização e a virtualização assistida por *hardware*. A virtualização completa realiza total abstração do sistema físico, criando um sistema físico virtual completo,

Figura 2 – Arquitetura de Virtualização de Servidores

Fonte: autoria própria

sobre o qual o sistema operacional hóspede é executado. A principal desvantagem é o desempenho, pois o *hypervisor* verifica a execução de todas as instruções privilegiadas ou sensíveis feitas pelo sistema operacional hóspede, e as substitui por ações equivalentes controladas (VERAS; KASSICK, 2011).

A paravirtualização é uma alternativa para melhorar os problemas de desempenho da virtualização completa. Na paravirtualização, o sistema operacional hóspede é alterado para chamar o *hypervisor* sempre que for executar uma instrução privilegiada ou sensível. A principal desvantagem da paravirtualização é a necessidade de modificação do sistema operacional hóspede, o que depende de acesso ao código fonte (VERAS; KASSICK, 2011).

Percebe-se que há uma relação custo benefício entre a virtualização completa e a paravirtualização. Enquanto a virtualização completa permite o uso de um sistema operacional hóspede sem alterações, a paravirtualização necessita alterá-lo para substituir instruções privilegiadas e sensíveis por *hypercalls*, mas oferece um melhor desempenho. Buscando melhorar essa relação de custo benefício, os fabricantes Intel e AMD investiram em extensões na arquitetura x86, para suportar a virtualização e melhorar o desempenho da solução como um todo. Essas extensões são chamadas de HAV (*Hardware Assisted Virtualization*). Dessa forma, a virtualização assistida por *hardware*, praticamente eliminou as vantagens de desempenho dos sistemas baseados em paravirtualização, que tinham o ônus de modificar o sistema operacional (VERAS; KASSICK, 2011).

Apesar da virtualização de servidores melhorar a utilização dos recursos e o consumo energético, ela também gera uma sobrecarga no sistema, independentemente do tipo de técnica empregada. Desta forma, a economia de energia não é proporcional à quantidade de VMs (CIMA et al., 2015).

2.2.1 Alocação de VMs

Além da sobrecarga no sistema, outro problema da virtualização de servidores é a alocação de VMs nas máquinas físicas. Alocar VMs, de uma forma eficiente pode não ser tarefa simples. Um dos algoritmos mais tradicionais, que pode ser aplicado para realizar esta tarefa, é o algoritmo FF (*First Fit*) (VARASTEH; GOUDARZI, 2015). Pode-se considerá-lo um algoritmo trivial, ele atribui um peso para cada VM, de acordo com os recursos demandados, e os servidores recebem a quantidade de pesos que conseguem hospedar. Assim, o algoritmo aloca cada uma das VMs no primeiro servidor físico com capacidade de hospedagem, até ter alocado todas as VMs. Caso não haja servidor com capacidade, um novo servidor deverá ser provisionado (VARASTEH; GOUDARZI, 2015).

Outro algoritmo que pode ser utilizado na alocação de VMs e que é uma variação do FF, é o FFD (*First Fit Decreasing*) (VARASTEH; GOUDARZI, 2015). A diferença é que ele organiza as VMs de forma decresente. A VM que demandar mais recursos, fica no topo de pilha e a que demandar menos recursos fica na base da pilha. Em seguida, o FFD vai alocando a VM do topo da pilha no primeiro servidor com capacidade de hospedagem, até ter alocado todas as VMs. Caso não exista servidor disponível, um novo servidor deverá ser provisionado (VARASTEH; GOUDARZI, 2015).

Apesar de serem algoritmos simples e funcionais, em certos casos eles não propiciam a solução ideal. Como exemplo, pode-se citar a seguinte situação (AMS, 2018):

- Existem seis VMs que precisam ser alocadas em dois servidores;
- As VMs possuem os seguintes pesos: 4, 8, 7, 10, 3 e 8, totalizando 40 pesos;
- Cada um dos servidores tem capacidade de hospedar 20 pesos.

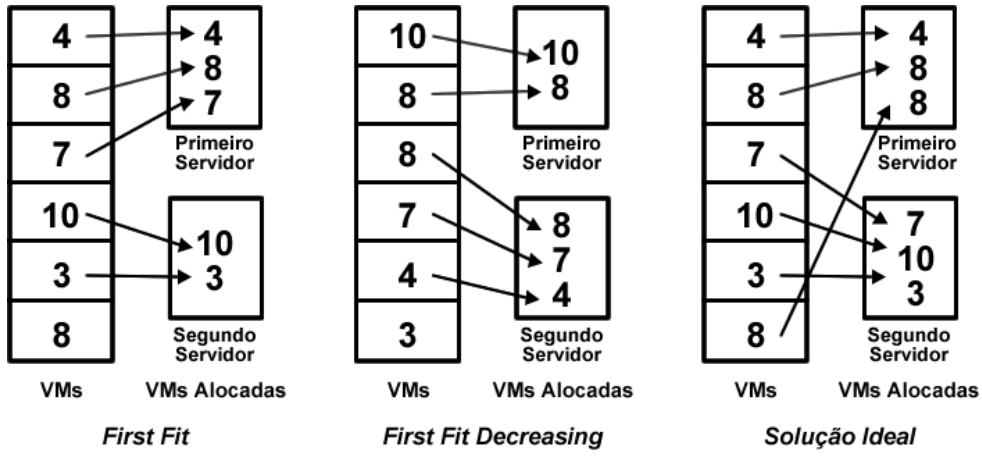
Utilizando o algoritmo FF, alocam-se as VMs de peso 4, 8 e 7 no primeiro servidor, as três VMs correspondem a 19 pesos. Na sequência, como as VMs de peso 10 e 3 não cabem no primeiro servidor, elas são alocadas no segundo servidor, as duas VMs correspondem a 13 pesos. A última VM a ser alocada possui peso 8, porém, o primeiro servidor possui 1 peso disponível e o segundo possui 7 pesos disponíveis. Dessa forma, ela não pode ser alocada em nenhum dos dois servidores, sendo necessário provisionar um terceiro servidor.

Aplicando o algoritmo FFD, primeiramente as VMs são organizadas da seguinte forma: 10, 8, 8, 7, 4, 3. Assim, alocam-se as VMs de peso 10 e 8 no primeiro servidor, o que corresponde a 18 pesos. Na sequência, como as VMs de peso 8, 7 e 4 não cabem no primeiro servidor,

elas são alocadas no segundo servidor, as três VMs correspondem a 19 pesos. A última VM a ser alocada possui peso 3, no entanto, o primeiro servidor possui 2 pesos disponíveis e o segundo 1 peso disponível. Assim, ela não pode ser alocada em nenhum dos dois servidores, sendo necessário o provisionamento de outro servidor.

Independentemente do algoritmo, não há uma alocação ideal. Apesar dos dois servidores possuírem capacidade de hospedar as seis VMs, há necessidade de provisionar um terceiro servidor. É importante salientar que existem outros algoritmos que podem ser empregados na alocação de VMs. Cada um com sua vantagem e desvantagem, como exemplo pode-se citar: NF (*Next-Fit*) e BF (*Best-Fit*) (JOHANSSON; AXELSSON; GUSTAVSSON, 2017). A Figura 3 mostra a alocação das VMs, do exemplo citado, utilizando os algoritmos FF e FFD e a solução ideal.

Figura 3 – Alocação de VMs Utilizando os Algoritmos FF, FFD e a Solução Ideal



Fonte: autoria própria

2.3 Conteinerização de Software

A virtualização de servidores, apesar de ser uma técnica importante, apresenta um impacto considerável no desempenho do sistema devido à sobrecarga gerada. Dessa forma, a conteinerização de *software* tem ganhado atenção (HEIDARI; LEMIEUX; SHAMI, 2016).

Contêiner é um termo muito utilizado na indústria naval moderna. Antes, não existia um padrão para o tamanho dos recipientes que transportavam os produtos nos navios. Isso gerava problemas, pois era difícil acomodar todos os recipientes de forma eficiente. Com a criação dos contêineres, a indústria naval padronizou o tamanho desses recipientes, e assim tem-se a

certeza de que haverá uma acomodação eficiente dos produtos nos navios. Esta é a mesma ideia da conteinerização de *software* na computação (DOCKER, 2017).

Para realizar a conteinerização de *software*, o *kernel* do sistema operacional hospedeiro deve ter suporte para essa função. Desta forma, tem-se um gerenciamento adequado dos recursos subjacentes e isolamento das atividades dos contêineres. Em vez de virtualizar o *hardware* completo, como é feito na virtualização de servidores, empacota-se a aplicação e suas dependências em um contêiner (BABU et al., 2014).

Assim, pode-se dizer que os contêineres são muito otimizados quando comparados com as VMs, e compartilham o *kernel* do sistema operacional hospedeiro e não há necessidade de criar um sistema físico virtual completo. Apesar de serem otimizados, cada contêiner conta com, além das aplicações e suas dependências, endereço IP (*Internet Protocol*), memória RAM (*Random Access Memory*) e pontos de montagens. Esse empacotamento provê uma portabilidade difícil de ser conseguida por outra plataforma (BABU et al., 2014), (CACCIATORE et al., 2015).

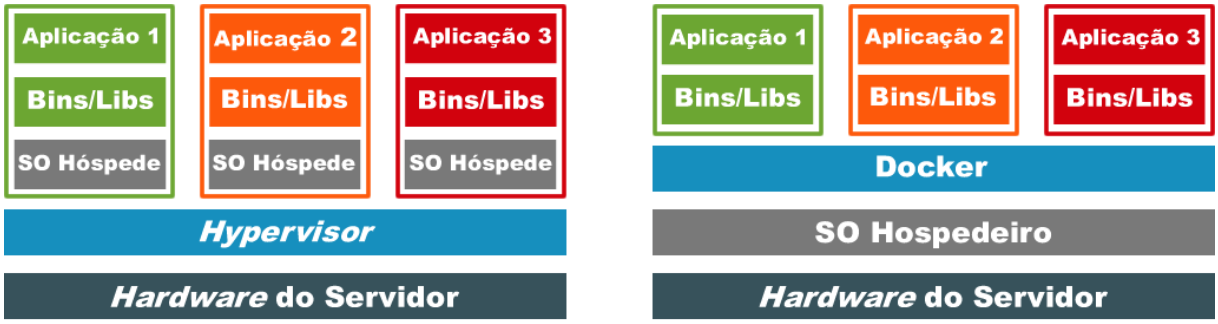
Além de gerar menos sobrecarga no sistema e possuir uma portabilidade considerável, a conteinerização de *software* melhora a elasticidade do sistema como um todo. Os contêineres iniciam muito mais rápido quando comparados com a iniciação de uma VM. Isso facilita o aumento ou diminuição de recursos provisionados de acordo com a carga de trabalho (BABU et al., 2014), (MANU et al., 2016).

O LXC (Linux Containers) é um conjunto de ferramentas, modelos e bibliotecas, que possibilita o gerenciamento de contêineres no Linux. Ele utiliza a propriedade *namespace* do *kernel* (REDHAT, 2019). E isso possibilita abstrair os recursos globais, fazendo com que cada processo dentro de um *namespace* comporte-se como se estivesse dentro de um sistema isolado. Alterações nos recursos de um contêiner são visíveis para outros processos que são membros do mesmo *namespace*, mas são invisíveis para outros processos. Assim, é possível gerenciar pontos de montagens, processos, rede, usuários e grupos dentro de um contêiner sem afetar os outros contêineres (CANONICAL, 2017).

Já o Docker é uma plataforma de código aberto que fornece uma maneira mais rápida de automatizar a criação e gerenciamento de contêineres. Ele basicamente melhora a usabilidade do LXC. Os contêineres Docker são criados usando imagens base. Uma imagem Docker pode conter apenas partes fundamentais de um sistema operacional, ou incluir também uma aplicação pré-configurada (BERNSTEIN, 2014).

A Figura 4 mostra a arquitetura de virtualização de servidores e de conteinerização de *software*. Percebe-se que na virtualização de servidores, além do *hypervisor*, existe um sistema operacional hóspede para cada aplicação. Já na conteinerização de *software* as aplicações e suas dependências são empacotadas pelo Docker e executadas pelo sistema operacional hospedeiro.

Figura 4 – Virtualização de Servidores x Conteinerização de Software



Fonte: adaptada de (COLEMAN, 2018)

2.4 Computação em Nuvem

A computação em nuvem é um paradigma que busca facilitar o acesso ubíquo, conveniente e sob demanda a recursos computacionais. Como exemplo de recursos computacionais, pode-se citar: redes, servidores, dispositivos de armazenamento e aplicações. Esses recursos devem ser provisionados rapidamente, com esforço mínimo e poucas interações entre o usuário e o provedor do serviço. Além disso, deve ser possível mensurar a utilização dos recursos para uma cobrança correta (WAN et al., 2016).

Existem três modelos básicos de serviços na computação em nuvem. Eles são descritos a seguir (LEHRIG; EIKERLING; BECKER, 2015):

- ***Infrastructure as a Service* (IaaS):** nesse modelo o provedor fornece os recursos de infraestrutura como serviço. Os usuários são responsáveis pela configuração e manutenção das VMs, sistemas operacionais e aplicações. A virtualização é um dos componentes centrais desse modelo (AHMAD; KANWAL; SHIBLI, 2013).

- ***Software as a Service* (SaaS):** nesse modelo, é fornecido a aplicação final aos usuários. O provedor é responsável pela implantação, configuração e manutenção de toda pilha, formada pela infraestrutura física, sistema operacional e aplicação. O controle administrativo do usuário é limitado.

- ***Platform as a Service* (PaaS):** o provedor é responsável por fornecer o ambiente de desenvolvimento. Esse ambiente, geralmente é composto por sistema operacional ou contêiner, compiladores, bibliotecas e sistema gerenciador de banco de dados. O usuário é responsável por desenvolver e hospedar suas aplicações.

Além da classificação por tipo de serviço oferecido, existem os modelos de implantação. Esses modelos definem quais serão os usuários dos serviços. As nuvens, de acordo com o modelo de implantação, são mostradas a seguir (WAN et al., 2016):

- **Pública:** fornece serviços para qualquer usuário interessado;
- **Privada:** para uso exclusivo dos usuários de uma organização;
- **Híbrida:** é a combinação da nuvem pública e privada. As nuvens, pública e privada, funcionam de maneira independente e conectam-se quando necessário.
- **Comunitária:** fornece serviços para um grupo de organizações.

2.5 OpenStack

Nos últimos anos, as plataformas de nuvem, como Eucalyptus, CloudStack e OpenStack, que fornecem a IaaS, ganharam relevância (GANGADHARAN, 2017). Entre elas, o OpenStack tem se destacado dos demais por conta da sua confiabilidade, elasticidade e robustez no controle de grandes quantidades de recursos computacionais (SHAO et al., 2015).

Atualmente, o OpenStack é uma das plataformas mais utilizadas nos *data centers*. É utilizado para criação de nuvens computacionais IaaS, públicas ou privadas. Foi criado em 2011 pela NASA e Rackspace, e hoje é apoiado por mais de 850 organizações (OPENSTACK, 2017d). Ele tem uma estrutura modular, escalável e flexível que garante a possibilidade de instalação em vários tipos de *hardware* (SHAO et al., 2015), (KOMINOS; SEYVET; VANDIKAS, 2017).

O OpenStack consiste em uma série de projetos inter-relacionados que possibilitam o controle de processamento, armazenamento e de recursos de rede em um *data center*. É possível gerenciá-lo através de uma aplicação *Web*, linha de comando ou através de uma API RESTful (*Representational State Transfer*) (WAN et al., 2016).

2.5.1 Arquitetura do OpenStack

A plataforma foi projetada para proporcionar, entre outras características, escalabilidade e elasticidade (OPENSTACK, 2017c). Escalabilidade, neste contexto, pode ser definida como facilidade de expansão dos recursos físicos de um *data center*, viabilizando assim o aumento do poder de processamento e armazenamento de dados. Já a elasticidade, pode ser definida como a capacidade do usuário aumentar ou diminuir de forma simples e rápida a quantidade de recursos computacionais provisionados para ele. (LEHRIG; EIKERLING; BECKER, 2015). Para conseguir tais características, o OpenStack é estruturado em tipos de servidores. Esses tipos são descritos a seguir (OPENSTACK, 2017c).

Servidores

Existem 4 tipos de servidores na arquitetura do OpenStack, dois deles são obrigatórios e dois opcionais (OPENSTACK, 2017c).

- **Servidor Controlador:** esse servidor é responsável por controlar os demais servidores da nuvem, assim ele é obrigatório. Ele controla todos os recursos disponíveis, desde VMs até rede e compartilhamentos.

- **Servidor de Computação:** assim como o servidor controlador, esse servidor também é obrigatório. Ele é responsável pelo processamento propriamente dito. Em uma instalação padrão, o servidor de computação executa o *hypervisor* KVM. Esse servidor também executa um agente de serviço de rede, que conecta as VMs às redes virtuais. Pode-se implantar mais de um servidor de computação. Quanto maior o número de servidores desse tipo, maior o poder de processamento da nuvem.

- **Servidor de Armazenamento em Blocos:** esse servidor é opcional. Ele contém os discos que os módulos de armazenamento em bloco fornecem para as VMs. Pode-se implantar mais de um servidor de armazenamento em bloco.

- **Servidor de Armazenamento de Objetos:** o servidor de armazenamento de objetos, assim como o servidor de armazenamento em blocos, também é opcional. Esse servidor, caso seja implantando, contém os discos utilizados para armazenar objetos.

O OpenStack é muito flexível, assim, dependendo da aplicação, pode-se implementar somente o servidor controlador e um servidor de computação (OPENSTACK, 2017c). A seguir

são apresentados os principais módulos que são instalados nos servidores (SAHASRABUDHE; SONAWANI, 2014).

Módulos

O OpenStack permite a instalação de módulos de acordo com os pré-requisitos de cada projeto. Porém, existem alguns que são obrigatórios. A seguir são apresentados os módulos mais importantes (OPENSTACK, 2017c).

- **Swift (*Object Storage*):** armazena e recupera objetos de dados não estruturados através de uma interface RESTful (*Representational State Transfer*) baseada em HTTP (*Hypertext Transfer Protocol*). Como exemplo de dados não estruturados, pode-se citar: imagens, vídeos, e-mails, documentos e binários. O Swift é muito escalar e tolerante a falhas;

- **Cinder (*Block Storage*):** foi projetado para ser um serviço de armazenamento de arquivos para usuários finais. Fornece uma API que possibilita solicitar e consumir esses recursos com o mínimo de conhecimento. O usuário não precisa se preocupar onde o dispositivo de armazenamento realmente está ou qual é o tipo de dispositivo. Também pode ser utilizado pelas instâncias de VMs em execução;

- **Glance (*Image Service*):** armazena e recupera imagens das máquinas virtuais, que são utilizadas durante o provisionamento de instâncias.

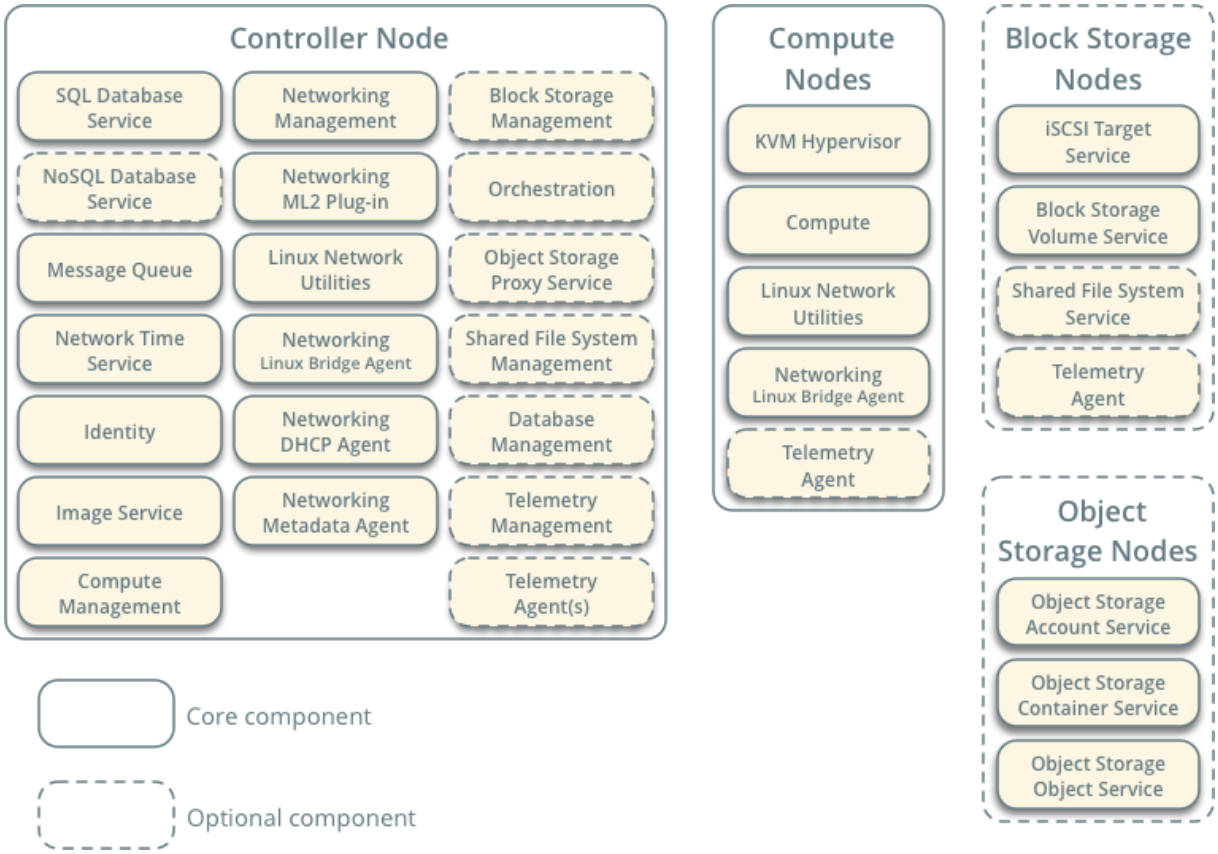
- **Neutron (*Networking*):** responsável por fornecer rede como um serviço para os demais módulos do OpenStack.

- **Nova (*Compute*):** gerencia o ciclo de vida das instâncias. Suporta uma grande variedade de *hypervisors*, como exemplo, pode-se citar: KVM, Xen, Hyper-V e VMware.

- **Keystone (*Identity*):** serviço de identidade usado pelos outros módulos para autenticação e autorização de alto nível. Atualmente, oferece suporte a autenticação baseada em *token* e usuário.

A Figura 5 apresenta uma visão geral da arquitetura do OpenStack. Nela, são mostrados os tipos de servidores e os módulos que compõem cada um desses servidores. Os servidores e os módulos obrigatórios, estão contornados por uma linha contínua. Já os servidores e módulos opcionais estão contornados por uma linha tracejada.

Figura 5 – Visão Geral da Arquitetura do OpenStack



Fonte: (OPENSTACK, 2017c)

Outros Modos de Implantação dos Servidores de Computação

Em uma instalação padrão, do OpenStack, instala-se o *hypervisor* KVM para a virtualização de servidores. Porém, é possível escolher outras duas formas de implantação deste servidor (KOMINOS; SEYVET; VANDIKAS, 2017), como descrito a seguir.

- **Bare-metal:** neste modo não há virtualização. A máquina física fica totalmente dedicada às aplicações e não há isolamento entre elas. O módulo Ironic é que provê esta funcionalidade ao OpenStack (OPENSTACK, 2017a).

- ***Contêineres***: este é um modo que consegue isolar recursos como na virtualização de servidores, porém com menos *overhead*. O módulo que possibilita esta funcionalidade é o Nova-docker. Dessa forma, é possível fornecer contêineres Docker para os usuários de uma nuvem OpenStack (KOMINOS; SEYVET; VANDIKAS, 2017).

2.6 Orquestração na Computação em Nuvem

A computação em nuvem é um paradigma que vem se consolidando. Empresas como Microsoft e Amazon Web Services são as principais construtoras desse modelo/paradigma. Porém, fazem-se necessárias intervenções de instituições a fim de trazer a padronização de uma variedade de linguagens, protocolos e arquiteturas comerciais. Atualmente, há uma necessidade de padronização da composição e automação dos serviços em nuvem. É neste cenário que o termo orquestração vem se destacando, sendo um dos tópicos mais estudados pelos pesquisadores. Apesar de não estar totalmente claro o que é a orquestração na computação em nuvem, ela geralmente trata sobre a composição e automação dos serviços (MOSCATO, 2015).

Desta forma, orquestração pode ser pensada como: resultado de uma automação inteligente, que proporcione o cumprimento do SLA (*Service Level Agreement*) e otimização dos recursos computacionais. Essa definição de orquestração pode ser inserida no contexto da arquitetura orientada a serviços, virtualização, provisionamento, infraestrutura convergente e tópicos de *data center* dinâmicos. Como exemplo, pode-se citar que a orquestração reduz o tempo e o esforço para ligar um novo servidor em uma nuvem computacional, fazendo com que, dependendo das necessidades, a nuvem ganhe poder de processamento.

Segundo Moscato (2015), algumas questões importantes sobre a orquestração são:

- Os provedores são os responsáveis pela gestão e execução da orquestração;
- A orquestração envolve os componentes e automação dos serviços em todos os níveis da pilha da nuvem (IaaS, PaaS e SaaS);
- Ela deve lidar com a qualidade, tanto dos componentes como dos serviços oferecidos.

2.7 Alocação Dinâmica de VMs

Grande parte dos estudos concentra-se no desenvolvimento de algoritmos de alocação dinâmica de VMs e desligamento de servidores subutilizados (ALI et al., 2015).

Um dos motivos para essa concentração de pesquisas, é porque a maioria dos servidores, mesmo quando ociosos, consomem cerca de 80% da potência total da fonte de alimentação. Desta forma, um servidor com carga de trabalho de apenas 15% da sua capacidade de processamento, pode consumir 80% da energia que consumiria se estivesse trabalhando no

seu limite. Além disso, em um *data center* típico, é comum existirem servidores subutilizados. Geralmente, esses servidores trabalham em uma faixa de 10 a 15% da sua capacidade de processamento, o que representa um desperdício considerável de energia (CIMA et al., 2015), (BARROSO; CLIDARAS; HöLZLE, 2013).

Foi observado por Orgerie, Lefevre e Gelas (2008) que os servidores consomem quase o limite da potência que a fonte de alimentação consegue fornecer, durante o *boot* do sistema. Além disso, observou-se que eles têm um consumo de energia considerável quando estão ociosos, corroborando os resultados apresentados em (CIMA et al., 2015). Dessa forma, considerando-se que o *boot* do sistema não é um processo tão demorado e que um servidor ocioso, pode ficar ligado por vários dias, a migração de VMs de um servidor com pouca carga de trabalho para outro servidor, e posteriormente o seu desligamento, pode ser vantajoso do ponto de vista da eficiência energética.

Assim, alocação dinâmica de VMs tem-se mostrado uma técnica importante. O que a torna mais atraente, é a possibilidade de migração a quente (VARASTEH; GOUDARZI, 2015). A migração a quente é um termo utilizado para a transferência de VMs entre servidores de computação sem ter que desligar a VM. O benefício de realizar este tipo de migração é que o serviço hospedado na VM a ser migrada, não sofre interrupção considerável. Dessa forma, o usuário final não é prejudicado (AWASTHI; GUPTA, 2016).

Além do benefício de não ser necessário interromper o serviço, existem outras vantagens da migração a quente. Entre elas, pode-se citar a manutenção do *hardware online* e economia de energia. Porém, este tipo de migração não traz somente vantagens, a migração a quente apresenta alguns riscos à segurança. Na maioria dos *hypervisors*, o principal risco é que os dados não são criptografados. O estado atual das aplicações em execução e dados sensíveis como senhas, por exemplo, são transmitidos em texto claro pela rede (UPADHYAY; LAKKADWALA, 2014). Para uma migração segura existe alguns requisitos, como: o servidor de origem e destino devem ser confiáveis, durante a transferência os dados devem permanecer confidenciais e deve haver mecanismos para detectar e relatar atividades suspeitas (AHMAD; KANWAL; SHIBLI, 2013).

A migração a quente envolve principalmente a transferência dos discos virtuais, dados dos registradores da CPU e conteúdo da RAM do servidor de origem para o servidor de destino. Existem duas abordagens básicas: uma é denominada pré-cópia e a outra pós-cópia. Na abordagem pré-cópia, o servidor de origem transfere o estado de memória da VM para o servidor de

destino em iterações. Na primeira iteração, o servidor de origem transfere toda a memória da VM para o servidor destino, nas iterações subsequentes, apenas as páginas que foram modificadas. Dessa forma, quanto mais ocorrerem modificações nas páginas de memória, mais tempo levará para a VM ser migrada. Assim, esta abordagem demora mais tempo para convergir, porém, é mais segura, tendo em vista que caso ocorra algum erro na rede, a VM continuará em execução no servidor de origem. (DESHPANDE; KEAHEY, 2015).

Na migração pós-cópia, a VM entra em execução no servidor de destino antes da transferência de suas páginas de memória. Apesar da pós-cópia diminuir o tempo de realocação da VM, ela provoca uma queda no desempenho das aplicações. Isso ocorre porque a aplicação é interrompida toda vez que uma página de memória não está disponível. A aplicação somente continua sua execução quando a página de memória é recuperada do servidor de origem (MAGALHAES; SOARES; GOMES, 2011).

Ambas as abordagens tradicionais têm suas vantagens e desvantagens. A pré-cópia não funciona bem em ambiente intensivo de escrita, o que causa um aumento importante no tempo total de migração. Por outro lado, a pós-cópia não funciona bem em ambiente intensivo de leitura, ocasionando uma degradação considerável no tempo de resposta da aplicação (SAHNI; VARMA, 2012).

Buscando equilibrar as vantagens e desvantagens, foi desenvolvida a abordagem híbrida. Nessa abordagem, combinam-se as técnicas da pré-cópia e pós-cópia. A migração híbrida é utilizada quando se deseja a confiabilidade da pré-cópia e velocidade de pós-cópia (SHAH; JAIKAR; NOH, 2015). Nessa abordagem, o servidor de origem realiza a primeira iteração onde é transferida toda a memória da VM para o servidor de destino. Durante a primeira iteração a VM continua sendo executada na origem, como na abordagem pré-cópia. Após o final da primeira iteração, a VM é colocada em execução no servidor de destino e, ao entrar em operação no servidor de destino, todas as páginas de memória que não estiverem disponíveis são buscadas no servidor de origem, assim como na pós-cópia (HINES; DESHPANDE; GOPALAN, 2009).

2.7.1 Parâmetros Considerados na Migração de VMs

Os algoritmos de alocação dinâmica de VMs consideram alguns parâmetros para decidir quais VMs migrar. Como exemplo, pode-se citar (VARASTEH; GOUDARZI, 2015):

- **Utilização do *hardware*:** esse é um dos parâmetros mais utilizados na consolidação de servidores. A utilização da CPU, memória, disco e rede é considerada para decisão de quais VMs migrar.

- **Dependência de dados:** é um parâmetro importante, onde o tráfego de rede e a comunicação entre as VMs são levados em consideração. Migrar VMs que comunicam entre si para um mesmo servidor, pode melhorar o desempenho do *data center* e a qualidade do serviço.

- **Confiabilidade do *hardware*:** outro parâmetro utilizado é a confiabilidade do *hardware*. A consolidação pode afetar e reduzir o tempo de vida dos servidores. Um servidor trabalhando quase no seu limite pode ter a vida útil reduzida. Migrar VMs sensíveis para servidores mais confiáveis pode ser vantajoso.

- **Custos de migração:** a utilização da largura de banda pela migração, pode ter efeitos negativos sobre a eficiência da rede para os usuários finais. Os algoritmos tendem a limitar as migrações simultâneas.

Além da alocação dinâmica de VMs entre os servidores de uma mesma nuvem computacional, outra alternativa é a criação de uma nuvem federada. Dessa forma, a alocação de VMs aconteceria entre os servidores de nuvens diferentes. Isso possibilita que certas nuvens sejam desligadas quase que totalmente em momentos de baixa carga de trabalho (WADHWA; VERMA, 2014), (CIMA et al., 2015).

A migração de VMs em uma nuvem federada pode ser aconselhável, porém isso nem sempre resulta em economia de energia. A vazão da rede WAN (*Wide Área Network*) não é o único obstáculo. As migrações podem, em alguns casos, ter um período de realização não aceitável devido ao tipo de carga de trabalho ou à ocorrência de falhas durante o processo. O que, neste último caso, exige que a migração seja reiniciada. Além disso, estudos apontam que durante a migração de VMs há um aumento no consumo de energia dos servidores envolvidos, de aproximadamente 10% em relação aos momentos sem migração (AIKEMA et al., 2012).

Criar VMs otimizadas e aumentar a largura de banda da rede, pode contribuir com a economia de energia. Isso ocorre, pois o tempo de migração é menor e os servidores ociosos seriam desligados mais rapidamente (DHANOA; KHURMI, 2015).

2.7.2 Migração de Contêineres

Os contêineres geralmente têm um tamanho muito menor que VMs, isso pode ser vantajoso na migração. Principalmente onde a vazão da rede, capacidade de armazenamento e processamento são limitados (KAUR et al., 2017). Porém, a migração de contêineres é uma área relativamente nova e que ainda não foi estudada amplamente. Dessa forma, existem algumas barreiras a serem vencidas. Uma delas é como implantar a migração de contêineres a quente. Atualmente as plataformas de contêineres não são totalmente estáveis nesse tipo de migração. Outra barreira é a compatibilidade entre as plataformas, contêineres são menos adaptáveis que VMs. Nos dias atuais, um contêiner Linux não pode ser executado em um servidor Windows (MACHEN et al., 2017).

Para realizar a migração, geralmente, os contêineres são desligados e posteriormente copiados para o servidor de destino. Se o contêiner estiver provendo algum serviço sem estado de conexão, as requisições que ocorrerem quando o contêiner estiver desligado serão afetadas. Porém nenhum trabalho anterior precisará ser refeito. Mas se o contêiner prover serviço orientado à conexão, o segmento será abortado e precisará ser requisitado novamente, aumentando assim o tempo de indisponibilidade (TAY; GAURAV; KARKUN, 2017).

Assim como na migração de VMs, os algoritmos de migração de contêineres consideram alguns parâmetros para decidir quais contêineres migrar. Esses parâmetros são praticamente os mesmos da migração de VMs.

Utilização da CPU, RAM e disco, largura de banda da rede e dependência de dados são alguns parâmetros citados por Piraghaj et al. (2015). Além disso, os autores Kaur et al. (2017) citam em seu trabalho que a abordagem mais simples para desencadear a migração de contêineres é baseada no consumo de energia da servidores. Desta forma, a migração é iniciada sempre que o consumo energético do servidor sair dos limites, inferior ou superior, pré-estabelecidos.

A migração pode ser um processo intensivo de I/O (*Input/Output*) que contribui para o aumento do consumo de energia elétrica em ambas as extremidades. Dessa forma, a migração de MVs ou contêineres, entre servidores de uma nuvem federada ou entre servidores de uma mesma nuvem requer estudos adicionais (MASTELIC et al., 2014). Outro ponto que merece atenção é o fato de que mesmo desligados, os servidores continuam consumindo cerca de 5% da potência total da fonte de alimentação (ORGERIE; LEFEVRE; GELAS, 2008).

2.7.3 Economia de Energia x SLA

A economia de energia, obtida por meio da alocação dinâmica de VMs, pode afetar a qualidade do serviço. Ela pode afetar tanto de forma positiva quanto negativa. Aplicações em servidores sobrecarregados, por exemplo, tendem a perder desempenho. Isso pode ocasionar violação do SLA. Projetar corretamente a capacidade máxima de VMs que um servidor suporta e alocar recursos suficientes para cada VM, pode minimizar estas violações (VARASTEH; GOUDARZI, 2015).

A disponibilidade de um sistema também afeta diretamente o SLA. A alta disponibilidade inclui a confiabilidade de componentes de *hardware* e de *software*. Sendo assim, um *data center* eficiente energeticamente pode prover mais qualidade de serviço do que outro menos eficiente energeticamente. No caso de interrupção do fornecimento de energia, por parte da concessionária de distribuição, o *nobreak* do *data center* mais eficiente, tende a mantê-lo ligado por mais tempo comparado com o *nobreak* do *data center* menos eficiente. Dessa forma, haveria menos interrupções do serviço hospedado no *data center* mais eficiente energeticamente. (SHARKH et al., 2015).

É oportuno definir o que significa dizer que um *data center* é mais eficiente energeticamente do que outro. Sendo assim, segue a definição de Corradi, Fanelli e Foschini (2014) e Barroso, Clidaras e Hölzle (2013): a capacidade de processamento de um *data center* está diretamente relacionada ao consumo de energia. Pois quanto maior é a capacidade de processamento, maior é a quantidade de equipamentos necessários e consequentemente o consumo de energia elétrica também é maior. Dessa forma, pode-se dizer que o *data center* A é mais eficiente do que o *data center* B, se A pode processar mais carga de trabalho do que B consumindo a mesma quantidade de energia elétrica, ou se A pode processar a mesma carga de trabalho que B, consumindo menos energia.

2.8 Projeto GEYSER

GEYSER (*Green nEtworked data centres as energY proSumers in smaRt city environments*) é um projeto da EU (*European Union*) que visa aumentar a eficiência energética dos *data centers* do futuro, implantados nas cidades inteligentes. Pensa-se que o gerenciamento da energia elétrica das cidades inteligentes será mais complexo do que hoje, tanto em termos de geração como de consumo, isso é o que impulsiona o projeto (UNION, 2017).

Pressupõe-se que as cidades inteligentes serão abastecidas por muitas fontes de energia e que uma porcentagem considerável desta será de fontes renováveis e gerada localmente. Dessa forma, o *data center* será um componente importante nesse contexto, e ele deverá ser capaz de gerar energia. Em momentos que o seu consumo for menor que a sua produção, ele deverá fornecer essa energia excedente para rede elétrica. Além disso, deverá ter capacidade de reduzir o seu consumo quando for necessário (CIMA et al., 2015).

Para conseguir tais requisitos, existem atualmente três trabalhos em desenvolvimento, que são listados a seguir (UNION, 2017):

- Sistema de controle avançado que possibilitará a visualização completa de todos os componentes do *data center*, incluindo a situação atual e perspectiva futura do sistema de energia elétrica;
- Sistema que possibilitará visualizar os recursos disponíveis, tanto de energia elétrica quanto de processamento e armazenamento de dados, em uma rede de *data centers* interconectados;
- Modelo de mercado local de energia que poderá ser usado para demonstrar como o *data center* reagiria a mudanças de condições no fornecimento de energia elétrica.

Com o decorrer do tempo e a maturidade do projeto, será possível realizar a orquestração da carga de trabalho em uma rede de *data centers*. Isso significa que a carga de trabalho da nuvem poderá ser adaptada com base nas condições de consumo e produção de energia elétrica. Se a energia estiver disponível a um preço baixo, o operador da nuvem poderá incentivar o uso dos recursos. Mas, se a energia estiver disponível a um preço elevado, o operador poderá incentivar a redução do uso (CIMA et al., 2015).

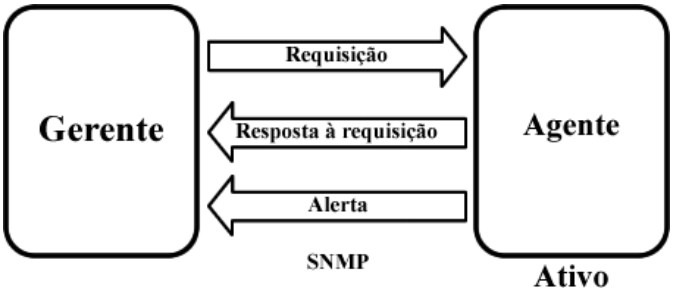
Além disso, no caso da energia com preço elevado, a carga de trabalho poderá ser deslocada no tempo ou no espaço. Deslocar a carga no tempo geralmente envolve o adiantamento do que pode ser processado em um momento em que há menor utilização dos recursos ou que a energia esteja a um preço mais baixo. Já no deslocamento no espaço, a carga de trabalho é movida para um *data center* de uma região que possui melhores condições de energia (CIMA et al., 2015).

2.9 SNMP

O SNMP é o protocolo mais utilizado para monitorar e gerenciar ativos de redes e serviços. O ativo a ser monitorado ou gerenciado deve possuir o agente SNMP. Assim, convencionou-se chamá-lo de agente. O componente/elemento que realiza consulta ou solicita modificações é denominado gerente. O SNMP é responsável por padronizar a comunicação entre o gerente e o agente, permitindo, assim, que uma ferramenta de gerenciamento possa trabalhar, de forma simples, com produtos e serviços de diversos fabricantes (CASE et al., 1990).

O agente, além de responder as requisições do gerente, também tem a função de gerar alertas. O gerente, por sua vez, pode usar esses alertas para realizar ações. Como exemplo de ação, pode-se citar o envio de *e-mails*, alertando sobre algum incidente envolvendo os ativos monitorados (CASE et al., 1990). A Figura 6 mostra os fluxos de comunicação entre o gerente e agente SNMP.

Figura 6 – Fluxos de Comunicação entre o Gerente e Agente SNMP



Fonte: (DUARTE et al., 2018)

O agente SNMP preenche uma tabela com informações que podem ser consultadas ou modificadas pelo gerente. Assim, é possível, por exemplo, consultar como está a utilização da CPU (*Central Processing Unit*) de um determinado servidor ou qual é a porcentagem de utilização da RAM (SONG; YOSHIHIRO; ASAMITOHRU, 2013).

Para que as consultas possam ser realizadas, o gerente precisa conhecer as informações que podem ser obtidas do agente. Isso é possível pelo uso de uma MIB (*Management Information Base*) e de um OID (*Object Identifier*). A MIB é base de informações de gerenciamento e o OID é o identificador único dentro da MIB. Assim o OID representa um ativo ou serviço dentro de uma hierarquia (MIKE, 2005).

Usualmente, o SNMP utiliza o protocolo UDP (*User Datagram Protocol*) nas portas 161 para os agentes e 162 para o gerente. O gerente envia solicitações na porta 161 do agente,

que por sua vez responde na porta 162 do gerente. O agente, além das respostas, também envia os alertas na porta 162 (CASE et al., 1990).

A combinação de vários agentes é chamada de comunidade SNMP. Cada comunidade SNMP é nomeada por uma *string*. A comunidade pode ser considerada uma forma de autenticação, pois um gerente somente consegue realizar requisições bem-sucedidas se souber qual é a comunidade dos agentes (CASE et al., 1990). A seguir são apresentadas as ferramentas, de gerenciamento SNMP, *Collectd* e *Sensu*. Estas são ferramentas, dentre outras, que facilitam o monitoramento utilizando o SNMP (FORSTER, 2018c), (SENSU, 2018c).

O *Collectd*, através de um *daemon* de monitoramento de recursos computacionais, coleta dados de desempenho de várias fontes. Dentre as fontes, pode-se citar o sistema operacional, aplicativos e arquivos de *log*. Esses dados podem ser utilizados para encontrar gargalos de desempenho e prever cargas futuras (FORSTER, 2018a). O *Collectd* tende a ser escalável, pois funciona no modo *push*. Além disso, por ser um *daemon* e permanecer na memória, tem um tempo de atualização padrão de dez segundos sem gerar sobrecarga considerável ao sistema (FORSTER, 2018b).

Já o *Sensu* é uma ferramenta que foca na compatibilidade e abrangência dos dispositivos e *software* monitorados. O *Sensu* possibilita o monitoramento de servidores, sistemas de gerenciamento de banco de dados, aplicações *Web*, ativos de rede, entre outros. Ele é um projeto de código aberto com suporte comercial (SENSU, 2018b). O *Sensu* possui dois *daemons*, um servidor e outro cliente. O servidor envia para os clientes o que deve ser monitorado e o tempo de atualização. Dessa forma, o cliente fica responsável por enviar os dados no tempo correto (SENSU, 2018a).

2.10 Wattímetros

Alguns trabalhos buscam contribuir com a redução do consumo energético, em *data centers*, através do monitoramento, uma vez que é difícil otimizar sem entender como é gasta a energia (ROSSIGNEUX et al., 2014), (MASTELIC et al., 2014), (KAVANAGH; ARMSTRONG; DJEMAME, 2016), (BARROSO; CLIDARAS; HöLZLE, 2013).

Existem vários monitores disponíveis, eles variam em termos de interface, modos de medições, atualização e exatidão. As principais interfaces são: ***(i)*** USB (Universal Serial Bus), ***(ii)*** Serial RS-232 (Recommend Standard - 232) e ***(iii)*** Ethernet (ROSSIGNEUX et al., 2014).

As conexões via portas seriais são utilizadas, geralmente, para conexão com os *no-breaks*. Para monitoramento individual, a conexão utilizada, geralmente é a *Ethernet*, em que podem ser transportados pacotes da IPMI (*Intelligent Platform Management Interface*) e SNMP (*Simple Network Management Protocol*) (ROSSIGNEUX et al., 2014).

Além disso, os monitores podem variar na maneira de operação. Os dois principais modos são: *push* e *pull*. No modo *push*, os equipamentos enviam as medições para o dispositivo central. Já no modo *pull* os servidores somente respondem às consultas realizadas pelo dispositivo central. A Tabela 1 mostra os principais dispositivos de monitoramento, suas interfaces, o tempo de atualização, medido em segundos, e a exatidão medida em W (Watts) (ROSSIGNEUX et al., 2014), (KAVANAGH; ARMSTRONG; DJEMAME, 2016).

Tabela 1 – Wattímetros

Dispositivo	**Interface**	**Tempo de Atualização (Segundos)**	**Exatidão ± (Watts)**
Dell iDrac6 [a]	Ethernet (IPMI)	5	7
Eaton	Serial e Ethernet(SNMP)	5	1
Omega Watt	Serial (IrDA)	1	0,125
Schleifenbauer	Ethernet (SNMP)	3	0,1
Watts Up?	USB (Proprietário)	1	0,1
ZEZ LMG450	Serial	0,05	0,001

[a] *Integrated Dell Remote Access Controller* versão 6

A escalabilidade é uma característica muitas vezes desejável em um sistema distribuído e, consequentemente, em um sistema de monitoramento do consumo de energia elétrica. Porém, normalmente ela entra em conflito com outras propriedades que também são desejáveis. Como exemplo, pode-se citar a confiabilidade. Quando a confiabilidade é desejável em um sistema distribuído, há necessidade de controle que detecte e retransmita as mensagens perdidas ou corrompidas. Isso envolve custos computacionais importantes em ambientes de larga escala. Até mesmo protocolos desenvolvidos especialmente para este fim, podem não se adaptar bem. Isso acontece devido à considerável quantidade de tráfego resultante das mensagens de controle (EUGSTER et al., 2003).

Desta forma, sob a ótica de custos computacionais, pode-se considerar que o monitoramento de apenas uma parte dos equipamentos é mais viável. Porém, impede-se que sejam capturados comportamentos importantes da infraestrutura. Outros trabalhos mostram que servidores com componentes desgastados podem ter um consumo de energia heterogêneo em um *data center* aparentemente homogêneo. Essa diferença pode chegar a 20%. Assim, do ponto

de vista da eficiência energética, é importante que um sistema possibilite o monitoramento de todos os equipamentos (ROSSIGNEUX et al., 2014).

2.10.1 IPMI

A IPMI é uma especificação de interface utilizada para gerenciar e monitorar o *hardware* de servidores. O seu desenvolvimento foi liderado pela Intel e hoje é suportado por mais de 200 fabricantes de *hardware* (INTEL et al., 2013). Seu funcionamento não depende de um sistema operacional específico, o que permite a administração ou monitoramento antes do sistema operacional ter sido iniciado. Pode-se monitorar sensores de temperatura do sistema, consumo energético, funcionamento de ventoinhas, funcionamento das fontes de alimentação e intrusão do chassi por exemplo. (INTEL et al., 2013). A seguir, são descritos alguns componentes importantes da IPMI.

Componentes da IPMI

- **BMC (*Baseboard Management Controller*):** é um dos principais componentes da arquitetura IPMI. É um microcontrolador ao qual todos os outros componentes se conectam por meio do IPMB (*Intelligent Platform Management Bus*). É o BMC que possibilita o monitoramento de forma independente do sistema principal (KOZAK; PREDKI; MAKOWSKI, 2011).

- ***Storage* SEL (*System Event Log*):** é um registrador não volátil de eventos do sistema e é gerenciado pelo BMC. Dessa forma, ele pode ser acessado através da LAN (*Local Area Network*), mesmo após uma falha da CPU do servidor. Como seu espaço é limitado, ele deve ser periodicamente verificado e excluído, de modo que eventos adicionais possam ser registrados/armazenados. Existem uma série de comandos IPMI que permitem ler e excluir registros do SEL (INTEL et al., 2013).

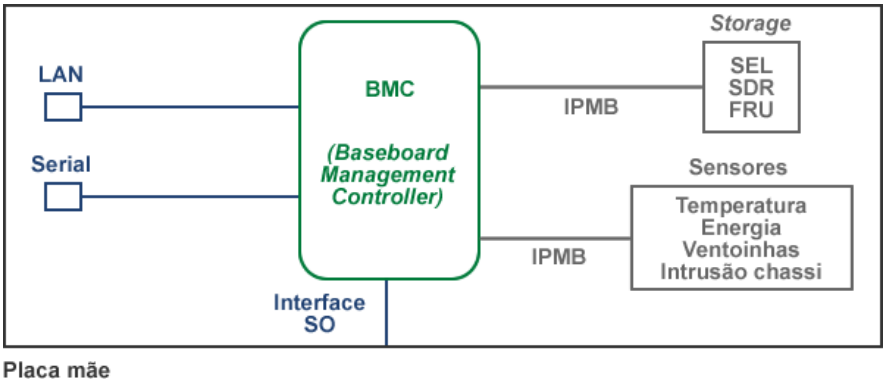
- ***Storage* SDR (*Sensor Data Record Repository*):** é responsável por armazenar, de forma não volátil, os dados coletados pelos sensores. Cada registro contém, além das informações sensoreadas, o tipo e o número do sensor (LEANGSUKSUN et al., 2006).

- ***Storage* FRU (*Field Replaceable Unit)*:** também é um registrador não volátil, responsável por armazenar dados sobre os *hardware*. Os principais dados armazenados são

número de série e modelo. Estes dados são utilizados para facilitar a aquisição de uma nova peça em caso de defeito (LEANGSUKSUN et al., 2006).

A IPMI permite comunicação, principalmente através da interface do sistema operacional, da interface serial e da LAN. Quando está sendo utilizada a LAN, as mensagens são transmitidas nos datagramas UDP e a porta de destino é a 623 (INTEL et al., 2013). A Figura 7 apresenta a arquitetura simplificada da IPMI.

Figura 7 – Diagrama Simplificado da Arquitetura IPMI



Fonte: autoria própria

Os sensores da IPMI não são tão precisos. O sensor que monitora o consumo energético, tem uma exatidão de ± 7 W (ROSSIGNEUX et al., 2014). Isso significa dizer, por exemplo, que uma oscilação no consumo dentro da faixa de 112 a 126 W não será detectada. Isso só pode ser resolvido pelo fabricante do hardware, que necessita realizar melhorias neste sensor. Apesar dessa exatidão, o sensor pode ser utilizado em vários cenários. Como exemplo, pode-se citar um ambiente onde não se deseja gerar carga adicional ao sistema principal ou onde deseja-se monitorar o consumo individual dos servidores (KAVANAGH; ARMSTRONG; DJEMAME, 2016).

2.10.2 Sonoff Pow

Sonoff Pow é um interruptor elétrico *smart*, sem fio (padrão 802.11 b/g/n), que possibilita ligar e desligar remotamente dispositivos eletrônicos conectados a ele, de forma agendada ou não. Além disso, o interruptor possibilita o monitoramento do consumo energético acumulado e instantâneo com uma exatidão de ±1%. O Sonoff Pow suporta trabalhar com diferença de potencial entre 90 e 250 Volts de corrente alternada. Já o limite de corrente é de 16 amperes e potência máxima de 3.500 Watts (ITEAD, 2018).

Por padrão, o equipamento vem com o *firmware* oficial do fabricante, que armazena os dados do monitoramento na nuvem computacional do próprio fabricante. Porém, ele possui uma interface física de 4 pinos que possibilita a troca do *firmware* original por outro. Dependendo do *firmware* escolhido, é possível alterar o local de armazenamento dos dados do monitoramento. Desta forma, pode-se aumentar as funcionalidades do equipamento.

Segundo o fabricante do Sonoff Pow (ITEAD, 2018), o equipamento apresenta uma melhor exatidão em relação ao wattímetro da IPMI (iDrac6). Porém, o wattímetro da IPMI já vem embarcado na maioria dos servidores. Assim, o Sonoff Pow representa um custo extra e, em *data centers* de larga escala, onde se deseja monitorar todos servidores, isso pode ser uma desvantagem a ser considerada. A Figura 8 mostra o Sonoff Pow com a abertura utilizada para conectar os cabos elétricos (ITEAD, 2018).

Figura 8 – Sonoff Pow

Fonte: (ITEAD, 2018)

2.11 Módulo de Telemetria do OpenStack

O OpenStack possui um módulo de monitoramento que é responsável por coletar e armazenar dados sobre a utilização dos recursos físicos e virtuais da nuvem. Atualmente, o projeto de telemetria fornece diversas funcionalidades divididas em vários subprojetos (OPENSTACK, 2017b).

O Ceilometer é um subprojeto do módulo de telemetria do OpenStack. Ele é responsável pela coleta de dados de eventos e medições de uma determinada nuvem implementada com o OpenStack. Isso possibilita o armazenamento em banco de dados ou encaminhamento para as filas de mensagens (OPENSTACK, 2017e).

Os dados coletados podem ser utilizados para vários fins. Como exemplo, pode-se citar o auxílio na solução de problemas com a nuvem. Além disso, esses dados coletados podem ser usados para contabilizar a utilização da nuvem por um determinado cliente, o que possibilita uma cobrança correta pela utilização. Já os dados enviados para as filas de mensagem podem ser utilizados, por exemplo, para emitir alarmes sobre o mal funcionamento dos principais módulos do OpenStack (OPENSTACK, 2017e).

2.12 Métricas de Avaliação de Desempenho de Redes

Avaliar o desempenho de redes de computadores pode ser importante para determinar a confiabilidade e/ou escalabilidade de um sistema. Para avaliar o desempenho de redes de computadores existem algumas métricas. Segundo Avramov (2019), como exemplo pode-se citar: ***(i)*** jitter, que é a variação estatística do atraso na entrega de dados, ***(ii)*** taxa de transferência, que é o número de bits por unidade de tempo encaminhado da origem para o destino menos a quantidade de bits perdidos e ***(iii)*** latência, que é a quantidade de tempo que leva entre a requisição e a resposta. Geralmente é medida em unidades de tempo, como segundos, milissegundos, microssegundos e assim por diante. Quanto menor for a latência, melhor é a qualidade da comunicação (AVRAM; SALEM; WONG, 2014). Além dessas métricas gerais, para avaliar o desempenho de servidores *Web*, existem métricas mais específicas. Como exemplo, pode-se citar: número de requisições atendidas e número de requisições não atendidas (APACHE, 2019b).

3 TRABALHOS RELACIONADOS

Há vários estudos que buscam reduzir o consumo de energia elétrica em *data centers*. A economia de energia nesses centros pode afetar a qualidade do serviço e diminuir custos (WADHWA; VERMA, 2014). Esta seção aborda alguns trabalhos relacionados. Primeiramente, discorre sobre o trabalho de Rossigneux et al. (2014), que apresenta uma API de monitoramento do consumo energético de nuvens computacionais implementadas com o OpenStack. Em seguida é apresentado o trabalho de Gandhi et al. (2012), que utiliza temporizadores para desligar servidores ociosos. Na sequência, apresenta-se o trabalho de Singh et al. (2013), que utiliza servidores temporários abastecidos com energia renovável. Após isso, discorre-se sobre o trabalho de Beloglazov e Buyya (2012) que emprega limites para a utilização da CPU como método de decisão para migrar VMs e desligar servidores ociosos. Por fim, apresenta-se o trabalho de Melhem et al. (2018), que apresentam um trabalho parecido com o dos autores Beloglazov e Buyya (2012), porém o algoritmo apresentado tenta prever a carga futura dos servidores, utilizando o modelo de predição de Markov.

KWAPI (KiloWatt API) é uma API relatada na literatura, que possibilita o monitoramento do consumo energético de nuvens, de grande escala, implementadas com o OpenStack. Além disso, esta API possibilita que os dados monitorados sejam integrados ao banco de dados do Ceilometer (ROSSIGNEUX et al., 2014), (CIMA et al., 2015).

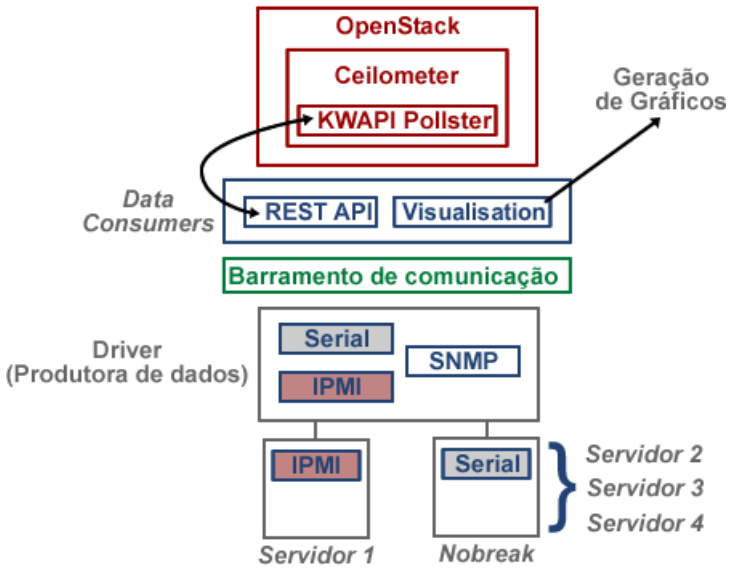
Segundo Rossigneux et al. (2014), ela foi desenvolvida com base em camadas e agentes, que são detalhados a seguir:

- **Driver:** é responsável pela medição propriamente dita do consumo de energia dos equipamentos. É considerada produtora de dados, pois fornece os dados coletados a camada *Data Consumers* através de um barramento de comunicação. Para realizar as medições, pode-se utilizar como interface a IPMI, Serial ou o protocolo SNMP.

- **Data Consumers:** esta camada possui dois agentes. O primeiro é chamado de API REST. Este agente recebe, da camada *driver*, o valor da tensão e da corrente de um determinado equipamento em um dado instante. Após isso, é calculado o consumo energético instantâneo e enviado ao agente KWAPI *Pollster*. O outro agente é chamado de *Visualisation*. Ele é responsável por gerar os gráficos, do consumo energético, que podem ser exibidos através de uma interface *Web*.

- **KWAPI Pollster:** é um agente utilizado para armazenar e recuperar dados no banco de dados do Ceilometer. A Figura 9 mostra a arquitetura da API.

Figura 9 – Arquitetura da KWAPI



Fonte: autoria própria

Cima et al. (2015), em seu trabalho, mostra que a KWAPI é uma API funcional. Ela foi utilizada para analisar o consumo de energia de uma determinada nuvem OpenStack. Os autores concluíram que servidores com pouca carga de trabalho tem um consumo de energia elevado, quando comparados com servidores com alta carga de trabalho. Desta forma, desligar servidores ociosos pode ser uma forma de alcançar a economia de energia. Isso condiz com outros trabalhos (DHANOA; KHURMI, 2015), (UPADHYAY; LAKKADWALA, 2014).

Além disso, a migração a quente híbrida tem uma eficiência aceitável para migrar VMs e posteriormente desligar os servidores ociosos. Os autores (CIMA et al., 2015) mostraram que o consumo energético de servidores é maior com carga intensiva de CPU do que com carga intensiva de disco rígido ou rede. Finalmente, relata-se que estão em andamento trabalhos sobre a realização de um gerente de migração de VMs para o OpenStack com base nessas conclusões.

Percebe-se que a KWAPI é uma API pioneira e que tem contribuído com a pesquisa e profissionais da área, porém ela apresenta um problema. O problema relatado no trabalho de Rossigneux et al. (2014) é que apesar da camada de *driver* gerenciar os incidentes, e também verificar periodicamente se todos os servidores estão ativos, a perda de uma medição pode ser significativa.

A KWAPI monitora somente o consumo energético instantâneo e utiliza estes dados para estimar o consumo acumulado. Assim, a quantidade de medições tem relação direta com

a exatidão desta estimativa. Portanto, a perda de uma amostra pode implicar em um cálculo impreciso do consumo energético acumulado. Além disso, ela é uma API específica para a plataforma OpenStack. A seguir são apresentadas algumas políticas dinâmicas para economia de energia em *data centers*.

Gandhi et al. (2012) afirma que políticas dinâmicas para economia de energia, em *data centers* com balanceamento de carga, podem ter um desempenho ruim por dois motivos. O primeiro é que estas políticas desligam muito rapidamente servidores considerados ociosos. O segundo motivo é a lentidão para ligar servidores que estejam desligados. Esses motivos podem resultar em consideráveis degradações de desempenho, principalmente em ambientes nos quais as cargas de trabalho oscilam significativamente.

Para mitigar esse problema, Gandhi et al. (2012) criaram um temporizador que evita o desligamento pré-maturo de servidores ociosos. Em momentos de baixa carga de trabalho, o balanceador de carga encaminha as requisições para um número reduzido de servidores. Nos demais servidores, é disparado um temporizador. Caso não ocorra picos de carga de trabalho, o temporizador dos servidores ociosos expira e assim são desligados. Caso ocorra picos de carga de trabalho, o balanceador de carga encaminha requisições para os servidores que estavam ociosos. Dessa forma, o temporizador é reiniciado e os servidores não são desligados. O grande problema desse mecanismo é encontrar o tempo ideal do temporizador. Um temporizador maior pode afetar a economia de energia, enquanto um menor pode afetar desempenho do *data center*.

Outra política para economia de energia, desta vez em *data centers* com virtualização, é a que utiliza servidores temporários. Nessa política, os servidores são classificados em dois tipos: estáveis e temporários. Os servidores estáveis, são servidores com fonte de energia confiável. Já os servidores classificados como temporários, são servidores que utilizam energia de fontes renováveis e que pode sofrer interrupção a qualquer momento. Em momentos que for possível produzir energia renovável, prioriza-se o processamento nos servidores temporários. Assim, economiza-se energia elétrica de fontes confiáveis, que geralmente é mais cara e poluente. Porém, como a energia dos servidores temporários pode acabar repentinamente, deve-se haver mecanismos que detecte antecipadamente essa interrupção. (SINGH et al., 2013).

Para tratar esse problema, em *data centers* com servidores temporários, os autores Singh et al. (2013) apresentam um mecanismo chamado Yank (puxão). O Yank monitora o *nobreak* para saber o estado do fornecimento de energia dos servidores temporários. Além disso, é mantido em um servidor estável, o *backup* das VMs que estão nos servidores temporários. O

Yank atualiza estas VMs de *backup* periodicamente, para manter a consistência entre elas e as VMs de operação. Esta medida visa, colocar as VMs de *backup* em operação rapidamente, em caso de interrupção da energia.

No caso de alerta de interrupção, o Yank realiza a última atualização das VMs de *backup* e as coloca em operação. Nos testes realizados, um único servidor estável de *backup* foi capaz de manter 15 VMs de servidores temporários com pouca degradação de desempenho. Após um alerta, o Yank leva em média apenas 10 segundos para colocar as VMs de *backup* em operação (SINGH et al., 2013). O grande problema desse mecanismo é se coincidirem uma interrupção de energia dos servidores temporários com um pico de carga de trabalho. Como as VMs estarão em apenas um servidor estável, provavelmente haverá importante degradação de desempenho.

Outro trabalho, para economia de energia em *data centers* com servidores virtualizados, é apresentado por Beloglazov e Buyya (2012). Os autores propõe uma política de alocação dinâmica de VMs, segundo a qual são estabelecidos um limite inferior e um limite superior de utilização das CPUs dos servidores. O propósito é manter a utilização das CPUs dentro da faixa estabelecida por essa política.

Quando um servidor estiver com a CPU abaixo do limite inferior ele será considerado como subutilizado, assim decide-se migrar todas as VMs deste servidor para outro servidor e na sequência coloca-se este servidor em modo de suspensão. Quando a CPU de um servidor estiver acima do limite superior, este deverá migrar algumas VMs para um servidor subutilizado. Apesar de apresentar bons resultados para os cenários testados, a validação ocorreu em simulador. Os autores relatam a importância de testar, a proposta apresentada, em ambiente real. Além disso, os autores utilizaram somente a carga da CPU para determinar se os servidores estavam subutilizados ou sobrecarregados. Conforme Varasteh e Goudarzi (2015) quanto maior o número de recursos monitorados, melhor será a alocação das VMs.

Outra política para economia de energia, relatada na literatura, é a dos autores Melhem et al. (2018). Os autores propõem um algoritmo parecido com o dos autores Beloglazov e Buyya (2012). A principal diferença é que o algoritmo proposto, tenta prever a carga futura dos servidores, utilizando o modelo de predição de Markov. O principal objetivo é evitar sobrecarregar os servidores e economizar energia. O algoritmo classifica os servidores com base na predição da utilização da CPU e nos limites inferior e superior.

As possíveis classificações são: ocioso caso a predição de utilização da CPU esteja abaixo do limite inferior, normal caso a predição esteja entre os limites inferior e superior, e

sobrecarregado caso a predição de utilização da CPU esteja acima do limite superior. Se o servidor for classificado como sobrecarregado ele é indicado para ter algumas VMs migradas para outros servidores. Porém, se servidor for classificado como ocioso, ele deverá ter todas as VMs migradas para outro servidor e posteriormente ser desligado.

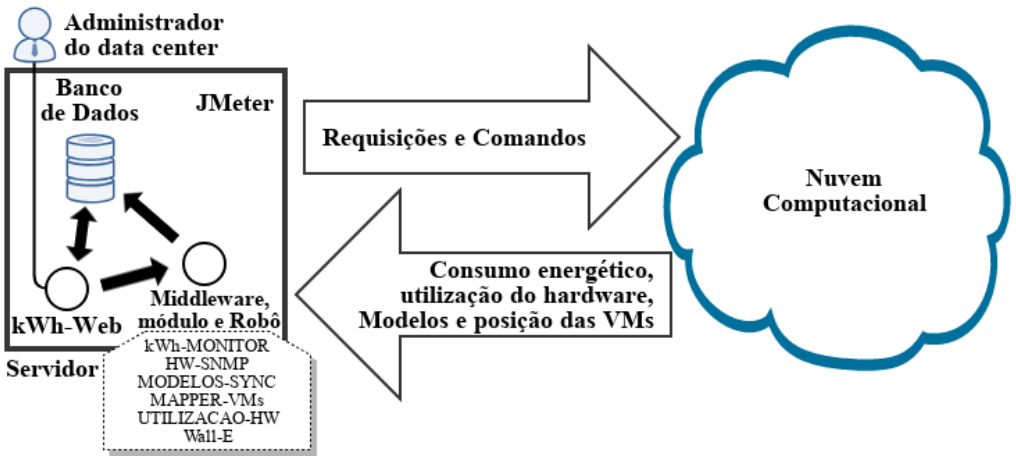
O algoritmo proposto pelos autores apresenta bons resultados, porém os autores levam em consideração somente a carga de trabalho da CPU. Isso pode ser um problema em ambientes com carga intensiva de I/O, por exemplo. Além disso, o algoritmo foi validado em simulador.

4 ARCABOUÇO KWH-PRO

Com o intuito de economizar energia elétrica em *data centers* com VMs, desenvolveu-se um arcabouço denominado kWh-PRO. O arcabouço possui: ***(i)*** quatro *middleware*, ***(ii)*** um módulo para calcular a utilização do *hardware*, ***(iii)*** um robô, ***(iv)*** uma aplicação *Web* e ***(v)*** um banco de dados. A seguir, são apresentados os *middleware* que compõem o arcabouço. O primeiro a ser apresentado, é o kWh-MONITOR, que é responsável pelo monitoramento do consumo energético dos servidores. O segundo é o HW-SNMP, que é responsável pelo monitoramento da utilização do *hardware* dos servidores. O terceiro é o MODELOS-SYNC, que é responsável pela sincronização dos modelos de VMs entre os bancos de dados do arcabouço e da nuvem computacional. O quarto *middleware* é o MAPPER-VMs, que é responsável pelo mapeamento da localização das VMs.

Após os *middleware*, apresenta-se o módulo que calcula a utilização do *hardware* dos servidores. Na sequência, apresenta-se o robô denominado Wall-E, que utiliza os dados coletados pelos *middleware* e o cálculo de utilização do *hardware* para decidir quais ações devem ser realizadas, tendo em vista a economia de energia e desempenho da nuvem. Por fim, é apresentada a aplicação *Web*, chamada de kWh-Web, que é responsável principalmente por facilitar a interação do administrador do *data center* com o banco de dados, além de gerar os parâmetros dos *middleware* e do robô. A Figura 10 apresenta, de forma abstrata, a arquitetura do arcabouço.

Figura 10 – Arquitetura do Arcabouço

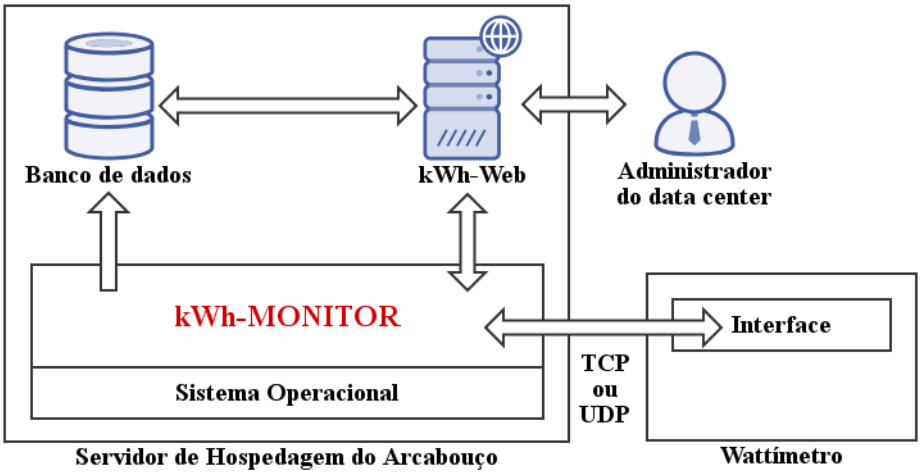


4.1 kWh-MONITOR - Monitor do Consumo Energético

O consumo energético dos servidores é monitorado pelo *middleware* kWh-MONITOR. Ele é uma interface entre wattímetros e banco de dados. Ele funciona no modo *pull*. Assim, en-

via requisições através da rede para os wattímetros, solicitando os dados de consumo energético. Os wattímetros, por sua vez, respondem a estas requisições. Assim, o *middleware* armazena, em banco de dados, os dados do consumo energético. A Figura 11 mostra, de forma abstrata, os fluxos de comunicação entre o kWh-MONITOR e os demais componentes.

Figura 11 – Fluxos de Comunicação entre o kWh-MONITOR e demais Componentes



kWh-MONITOR é descrito pelo Algoritmo 1. Ele contém um procedimento chamado *Medir*, que recebe como parâmetros: ***(i)*** endereço IP do wattímetro, ***(ii)*** credenciais para conexão, ***(iii)*** identificador único do servidor no banco de dados, e ***(iv)*** tipo de consumo, que pode ser instantâneo ou acumulado (linha 1). Com esses parâmetros, o procedimento solicita o consumo de energia de cada servidor (linhas 3 e 6). O procedimento organiza e concatena os resultados em uma variável (linhas 4 e 7). O monitoramento do consumo instantâneo é executado nas (linhas 10 a 12). O monitoramento do consumo acumulado ocorre a cada 10 minutos (linhas 13 a 17). Como o valor do consumo acumulado varia pouco ao longo do tempo, ele aumenta lentamente, então determinou-se empiricamente a coleta em intervalos de 10 minutos. No final de cada ciclo, os dados são inseridos no banco de dados (linha 18).

Algoritmo 1 kWh-MONITOR - Monitoramento do Consumo Energético

```
1 procedimento MEDIR(EnderecoIP, Credenciais, ID.servidor, TipoConsumo)
2     se (TipoConsumo = instantaneo) então
3         LeituraWattimetro ← ConsumoInstanteWattimetro(EnderecoIP, Credenciais);
4         Dados ← Dados + (ID.servidor, LeituraWattimetro, TipoConsumo, DataServidor)
5     senão
6         LeituraWattimetro ← ConsumoAcumuladoWattimetro(EnderecoIP, Credenciais);
7         Dados ← Dados + (ID.servidor, LeituraWattimetro, TipoConsumo, DataPeriodo)
8     fim se
9 fim procedimento
```

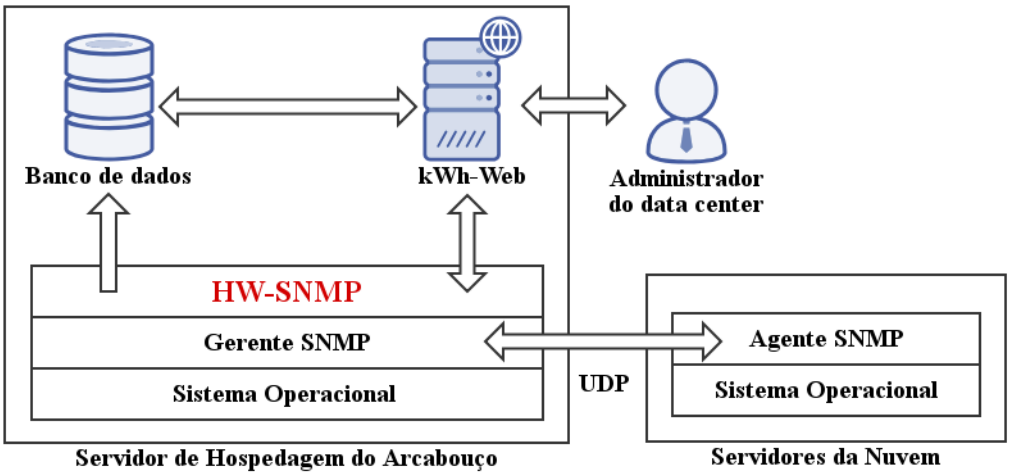
```
10 para (i ← 1; i <= NumeroNos; i++) faça
11 |   MEDIR(EnderecoIPNo.i, Credenciais.i, ID.servidor.i, instantaneo)
12 fim para
13 se (Minuto mod 10 = 0 ) então
14 |   para (i ← 1; i <= NumeroNos; i++) faça
15 |   |   MEDIR(EnderecoIPNo.i, Credenciais.i, ID.servidor.i, acumulado);
16 |   fim para
17 fim se
18 TabelaConsumo ← Dados;
```

A principal diferença entre esse *middleware* e outros é que ele permite o monitoramento do consumo energético instantâneo e acumulado. Dessa forma, caso ocorra a perda de alguma mensagem isso não influenciará de forma significativa no monitoramento, já que o consumo acumulado é estimado pelo próprio wattímetro monitorado e não pelo *middleware*. Além disso, trata-se de um *middleware* genérico, que não está integrado ao Ceilometer do OpenStack. Isso permite monitorar o consumo energético de *data centers*, com as mais variadas plataformas.

4.2 HW-SNMP - Monitor da Utilização do Hardware

Além do kWh-MONITOR, foi desenvolvido o *middleware*, denominado HW-SNMP. Que possibilita o monitoramento da utilização do *hardware* de servidores. É possível monitorar os seguintes dispositivos: ***(i)*** CPU e ***(ii)*** RAM. Esse *middleware* também funciona no modo *pull*. Ele utiliza a estrutura de gerente e agente do protocolo SNMP para requisitar os dados de utilização do *hardware*. Posteriormente, armazena-os em banco de dados. A Figura 12 apresenta os fluxos de comunicação entre o HW-SNMP e os demais componentes.

Figura 12 – Fluxos de Comunicação entre o HW-SNMP e demais Componentes



O HW-SNMP é descrito pelo Algoritmo 2. Ele contém um procedimento chamado *Monitorar*, que recebe como parâmetros: ***(i)*** endereço IP do sistema operacional do servidor, ***(ii)*** ID do servidor, e a ***(iii)*** comunidade SNMP que o servidor faz parte (linha 1). Com estes três parâmetros, é requisitado o tamanho total (linha 2) e o espaço livre da RAM (linha 3). Assim, com estes dados é possível calcular a porcentagem livre da RAM (linhas 4 a 6).

Na sequência são requisitadas as informações da CPU. Todas as informações são armazenadas em uma variável (linha 7). Assim, calcula-se a utilização global da CPU dividindo a soma da utilização dos núcleos pela quantidade de núcleos (linha 8). Por fim o procedimento concatena, de forma estruturada, os resultados de todas as chamadas (linhas 11 a 13) em uma variável (linha 9). Dessa forma, ao final de cada ciclo, é possível inserir os dados no banco (linha 14). Optou-se por não monitorar a utilização dos HDs dos servidores, pois os discos das VMs ficavam armazenados, de forma centralizada, em um servidor NFS.

Algoritmo 2 HW-SNMP - Monitor da Utilização do Hardware

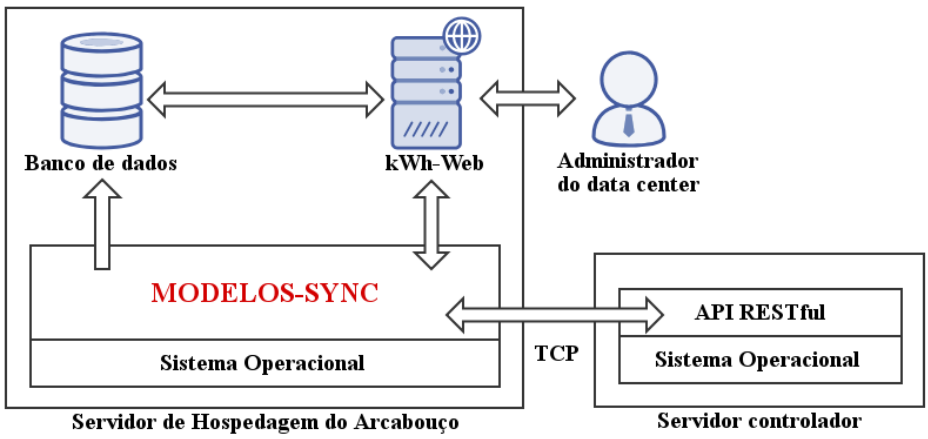
```
1  procedimento MONITORAR(EnderecoIP, ID.servidor, Comunidade)
2  |  TamanhoRAM ← (RequisicaoSNMP, Comunidade, EndIP, MIB1.OID1);
3  |  EspacoLivreRAM ← (RequisicaoSNMP, Comunidade, EndIP, MIB1.OID2);
4  |  Regra3 ← (EspacoLivreRAM * 100);
5  |  PorcLivreRAM ← (Regra3/TamanhoRAM);
6  |  UtilizacaoRAM ← (100 − PorcLivreRAM);
7  |  InformacoesCPU ← (RequisicaoSNMP, Comunidade, EndIP, MIB2.OID1);
8  |  UsoCPU ← (InformacoesCPU.usoNucleos/InformacoesCPU.QtdNucleos);
9  |  Dados ← Dados + (ID.servidor, UtilizacaoRAM, UsoCPU, DataServidor)
10 fim procedimento
11 para (i ← 1; i <= NumeroServidores; i++) faça
12 |  MONITORAR(EnderecoIPservidor.i, ID.servidor.i, Comunidade);
13 fim para
14 TabelaCargaTrabalho ← Dados;
```

4.3 MODELOS-SYNC - Sincronizador de Modelos de VM

Outro *middleware* do arcabouço é o MODELOS-SYNC. Ele é responsável por inserir no banco de dados do arcabouço os modelos de VMs criados através da plataforma de nuvem. Esta sincronização é importante para decidir qual VM migrar primeiro. Isso porque é com base no nome do modelo que as VMs são ordenadas. O arcabouço utiliza o algoritmo FFD, e para conseguir ordenar as VMs em ordem decrescente, durante as consultas SQL (*Structure Query Language*), o único pré-requisito é nomear os modelos de VMs em ordem alfabética, de acordo com a capacidade de cada um. O modelo que possui a maior configuração deve ficar em primeiro lugar na ordem alfabética e assim por diante.

Para realizar a sincronização, o MODELOS-SYNC autentica na nuvem e requisita os dados dos modelos de VMs através de API RESTful do OpenStack que o servidor controlador disponibiliza. Posteriormente, o MODELOS-SYNC armazena em variáveis o nome do modelo, a quantidade de CPU, RAM e de HD de cada um dos modelos e armazena no banco de dados do arcabouço. A Figura 13 apresenta os fluxos de comunicação entre o MODELOS-SYNC e os demais componentes.

Figura 13 – Fluxos de Comunicação entre o MODELOS-SYNC e demais Componentes



O MODELOS-SYNC é descrito pelo Algoritmo 3. Ele possui um procedimento chamado *SincronizaModelos* (linha 1). Após receber a chamada (linha 12), o procedimento requisita, ao servidor controlador, um *token* de autenticação. Para isso, há necessidade de utilizar credenciais de acesso da nuvem (usuário e senha) (linha 2). Com o *token*, são requisitados todos os modelos de VMs e a resposta é armazenada em variável (linha 3). Posteriormente, a variável é percorrida até o final, momento em que o nome, quantidade de CPU, de RAM e de HD, de cada um dos modelos, são armazenados em variáveis separadas (linhas 5 a 8). Dessa forma, o procedimento vai concatenando de forma estruturada, os dados em uma variável única (linha 9). Assim, ao final, é possível inserir os dados no banco de dados (linha 13).

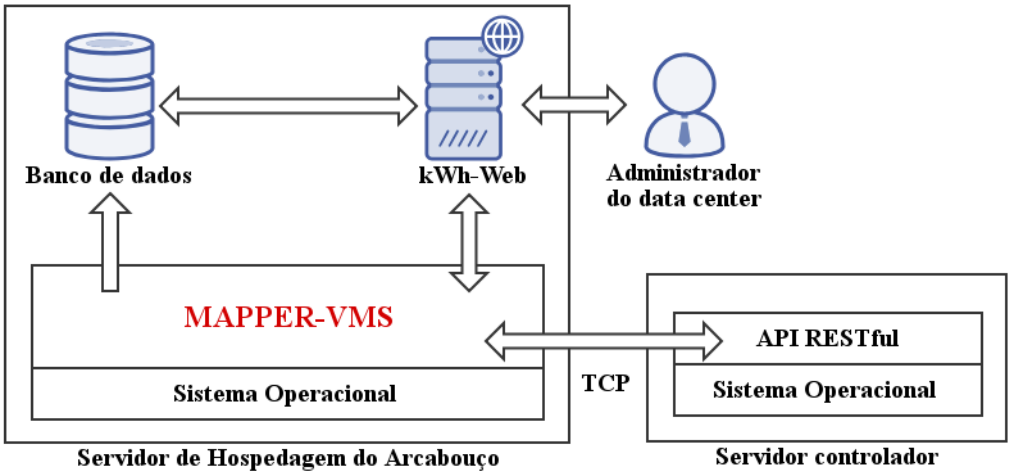
Algoritmo 3 MODELOS-SYNC - Sincronizador de Modelos de VM

```
1  procedimento SincronizaModelos()
2      Token ← [RequisitaAutenticacao → URL_Autenticacao(UsuarioNuvem, Senha)];
3      Modelos ← [RequisitaModelos → URL_API(Token, controlador)];
4      para (Modelos <> Fim) faça
5          NomeModelos ← Modelos.Nome;
6          QtdCPU ← Modelos.QtdCPU;
7          QtdRAM ← Modelos.QtdRAM;
8          QtdHD ← Modelos.QtdHD;
9          Dados ← Dados + (NomeModelos, QtdCPU, QtdRAM, QtdHD);
10     fim para
11 fim procedimento
12 Chamada → SincronizaModelos
13 TabelaModelosVM ← Dados;
```

4.4 MAPPER-VMs - Mapeador de Localização de VMs

O MAPPER-VMS é quarto e último *middleware* do arcabouço. Ele é responsável por identificar em qual servidor de computação cada uma das VMs está hospedada. Conhecer a localização de cada VM é essencial para realizar migrações. Para realizar este mapeamento, o MAPPER-VMs autentica na nuvem e requisita os dados das VMs através de API RESTful que o servidor controlador disponibiliza. Posteriormente, o MAPPER-VMs armazena os dados no banco de dados do arcabouço. A Figura 14 apresenta os fluxos de comunicação entre o MAPPER-VMs e os demais componentes.

Figura 14 – Fluxos de Comunicação entre o MAPPER-VMS e demais Componentes



O MAPPER-VMs é descrito pelo Algoritmo 4. Ele possui um procedimento chamado *MapeamentoVMs*, que recebe como parâmetro o nome do servidor de computação que deseja saber quais VMs ele hospeda (linha 1). Após o recebimento do parâmetro, um *token* de autenticação é requisitado ao servidor. Para isso, há necessidade de utilizar credenciais de acesso da nuvem (usuário e senha) (linha 2). Com o *token*, requisita-se ao servidor controlador, todas as VMs hospedadas no servidor de computação, a resposta é armazenada em variável (linha 3). Posteriormente, a variável é percorrida até o final, quando o ID da VM, o estado da VM (ligada ou desligada) e modelo, pelo qual a VM foi criada, são armazenados em variáveis separadas (linhas 5 a 7). Dessa forma, o procedimento vai concatenando de forma estruturada, os dados em uma variável única (linha 8). Assim, ao final, é possível inserir os dados no banco de dados (linha 14).

Algoritmo 4 MAPPER-VMs - Mapeador de Localização de VMs

```
1  procedimento MapeamentoVMs(NomeServidorComputacao)
2      Token ← [RequisitaAutenticacao → URL_Autenticacao(UsuarioNuvem, Senha)];
3      VMs ← [RequisitaVMs → URL_API(Token, controlador(NomeServidorComputacao))];
4      para (VMs <> Fim) faça
5          ID_VM ← VMs.ID;
6          Estado ← VMs.estado;
7          Modelo ← VMs.modelo;
8          Dados ← Dados + (ID_VMs, Estado, Modelo, Servidor);
9      fim para
10 fim procedimento
11 para (i ← 1; i <= NumeroServidores; i++) faça
12     MapeamentoVMs(NomeServidor);
13 fim para
14 TabelaVMs ← Dados;
```

4.5 UTILIZACAO-HW - Calculador de Utilização do *Hardware* dos Servidores

De acordo com Varasteh e Goudarzi (2015) quanto maior o número de recursos monitorados, melhor é a alocação dinâmica de VMs. Trabalhos como de Beloglazov e Buyya (2012) e Melhem et al. (2018) utilizam apenas a carga da CPU para decidir sobre migrações e desligamento de servidores. Avaliar somente a carga da CPU para decidir sobre migrações e desligamento dos servidores pode ser problemático. Tendo em vista que podem haver ambientes com carga intensiva de RAM. Assim, um servidor com carga de CPU intermediária, pode apresentar alta carga de RAM. Dessa forma, as decisões sobre as migrações e desligamento de servidores podem não ser as melhores.

Com o intuito de tratar este problema, elaborou-se o módulo do arcabouço denominado UTILIZACAO-HW, que é responsável por calcular a utilização do *hardware* de cada servidor de computação. O UTILIZACAO-HW utiliza uma fórmula para que seja possível expressar a utilização do *hardware* em apenas um valor numérico. Dessa forma, há possibilidade de tomar decisões com base na carga da CPU, RAM e limites definidos. Ele é descrito pelo Algoritmo 5, que possui um procedimento chamado *Calcular*. O procedimento recebe como parâmetros: ***(i)*** usuário do banco de dados, ***(ii)*** senha do usuário do banco de dados e ***(iii)*** ID do servidor de computação (linha 1).

Após receber os parâmetros, o procedimento conecta no banco de dados utilizando o usuário e senha (linha 2). Dessa forma, é possível utilizar a função do sistema de gerenciamento de banco de dados para consultar o valor médio de utilização da CPU e RAM. Para selecionar os dados corretos de cada servidor, é utilizada uma cláusula que condiciona a igualdade entre

campo ID.servidor e o ID recebido como parâmetro. O campo ID.servidor é a chave primária que identifica, de forma única, cada um dos servidores nas tabelas do banco de dados. Para cada consulta SQL, busca-se a média dos oito últimos valores monitorados de cada item. Para cada uma das consultas, o valor retornado é armazenado na variável correspondente: MediaCPU e MediaRAM. (linhas 3 e 4). Após consultar e armazenar em variáveis a média de utilização da CPU e RAM é calculada a utilização do *hardware*. Para este cálculo, utiliza-se a seguinte fórmula: $\textbf{UtilizacaoHardware} = \frac{\textbf{(MediaCPU + MediaRAM)}}{\textbf{2}}$.

Percebe-se que é realizada uma média aritmética simples, em que a utilização da CPU e RAM possuem o mesmo peso (linha 5). O procedimento vai concatenando, de forma estruturada, os resultados de todas as chamadas (linhas 8 a 10) em uma variável (linha 6). Dessa forma, ao final de cada ciclo, é possível inserir os dados no banco de dados (linha 11).

Algoritmo 5 UTILIZACAO-HW - Calculador de Utilização do Hardware dos Servidores

```
1  procedimento CALCULAR(Usuario, Senha, ID)
2  |  ConectaBancoDados(Usuario, Senha);
3  |  MediaCPU ← (FuncaoCalcMedia CPU, onde (ID.servidor = ID) Limite8, Decrescente);
4  |  MediaRAM ← (FuncaoCalcMedia RAM, onde (ID.servidor = ID) Limite8, Decrescente);
5  |  UtilizacaoHardware ← [(MediaCPU + MediaRAM)]/2
6  |  Dados ← Dados + (ID, MediaCPU, MediaRAM, UtilizacaoHardware)
7  fim procedimento
8  para (i ← 1; i <= NumeroServidores; i++) faça
9  |  CALCULAR(Usuario, Senha, ID.i);
10 fim para
11 TabelaPontuacao ← Dados;
```

4.6 Wall-E - Robô Balanceador de Carga e Economizador de Energia

O robô Wall-E é responsável por realizar ações sobre uma determinada nuvem computacional, tendo em vista a economia de energia e desempenho. Para isso, ele utiliza os dados coletados pelos *middleware* do kWh-PRO. Para cada tarefa realizada pelo Wall-E, existe um procedimento. As tarefas realizadas pelo robô são as seguintes: ***(i)*** classificar os servidores, ***(ii)*** migrar VMs entre os servidores, ***(iii)*** ligar e desligar servidores de computação, ***(iv)*** atualizar o estado dos servidores no banco de dados do arcabouço e ***(v)*** registrar as principais tarefas executadas por ele mesmo. Além disso, existe um procedimento principal, que decide qual tarefa realizar. Os procedimentos do Wall-E estão descritos nas próximas subseções.

O nome Wall-E é referência ao robô do filme de ficção científica de mesmo nome, produzido pela empresa Pixar Animation Studios. Wall-E, abreviação de *Waste Allocation Load*

Lifter Earth-class, é o último robô deixado na Terra, que passa o dia arrumando o lixo do planeta (MORRIS et al., 2018).

4.6.1 Classificação dos Servidores

O procedimento *Classificar* é responsável por realizar a classificação dos servidores de computação. É através dessa classificação que o Wall-E decide quais tarefas devem ser executadas. A classificação é realizada com base na: ***(i)*** utilização do *hardware*, calculada pelo módulo UTILIZACAO-HW, ***(ii)*** limites superior, de segurança e inferior, ***(iii)*** se há VM hospedada no servidor e ***(iv)*** estado do servidor (ligado ou desligado). As possíveis classificações são: ***(i)*** sobrecarregado, ***(ii)*** quase sobrecarregado, ***(iii)*** normal, ***(iv)*** quase ocioso, ***(v)*** ocioso e ***(vi)*** desligado.

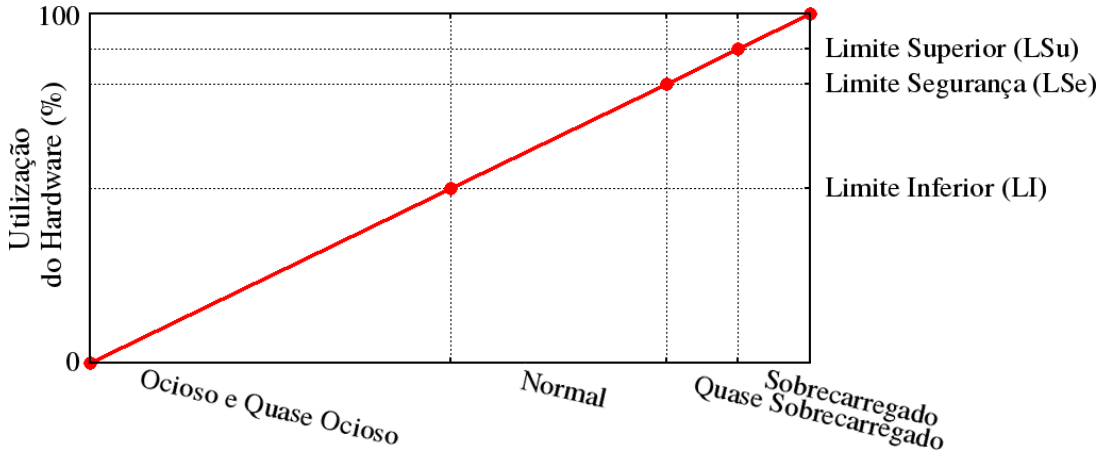
Limite superior e inferior são valores percentuais, cadastrados no banco de dados através da kWh-Web, que delimitam a faixa de operação dos servidores, tendo como bases a economia de energia elétrica e desempenho da nuvem. Porém, durante os experimentos iniciais, percebeu-se que somente esses limites não eram suficientes. Servidores que estavam com a utilização do *hardware* muito próxima do limite superior e recebiam mais VMs, muitas vezes passavam a ser considerados sobrecarregados. Isso causava migrações desnecessárias, sendo essencial criar um controle que diminuísse essas migrações.

O controle criado é o limite de segurança, que também é um valor percentual menor que o limite superior. Dessa forma, todos os servidores que estão com a utilização do *hardware* entre o limite de segurança e limite superior, são considerados quase sobrecarregados e não podem receber VMs de outros servidores. Esta faixa de classificação diminui consideravelmente as migrações desnecessárias. Essa é uma importante contribuição deste trabalho. Pois trabalhos como de Beloglazov e Buyya (2012) e Melhem et al. (2018) propõem somente três classificações para os servidores. São elas: ***(i)*** ociosos, ***(ii)*** normais e ***(iii)*** sobrecarregados.

De acordo com a política proposta pelos autores (BELOGLAZOV; BUYYA, 2012) e (MELHEM et al., 2018), quando um servidor é classificado como ocioso, e existe servidor classificado como normal, deve-se migrar todas as VMs do servidor ocioso para o servidor normal. Isso pode ser considerado um notável problema. Isso porque, se o servidor classificado como normal estiver próximo do limite superior, após receber todas as VMs do servidor ocioso, provavelmente será considerado sobrecarregado. Dessa forma, é necessário realizar novas migrações até o servidor deixar de ser classificado como sobrecarregado. Conforme Mastelic et al. (2014)

a migração pode ser um processo intensivo de I/O, que contribui para o aumento do consumo de energia em ambas extremidades. Além de que pode contribuir também com a degradação do desempenho da nuvem computacional. Dessa forma, é importante diminuir o número de migrações desnecessárias. A Figura 15 mostra as classificações dos servidores proposta por este trabalho, com base na utilização do *hardware* e nos limites.

Figura 15 – Classificações dos Servidores



O procedimento *Classificar* é descrito pelo Algoritmo 7, ele utiliza variáveis do tipo vetor para organizar os dados dos servidores de acordo com suas classificações. Os dados organizados são: ***(i)*** endereço IP do servidor, ***(ii)*** ID da primeira VM hospedada no servidor (FFD), ***(iii)*** o ID do servidor no banco de dados, ***(iv)*** o endereço IP da interface IPMI do servidor e ***(v)*** as credenciais de acesso da IPMI. Tanto o endereço IP como o ID da VM são utilizados em uma eventual migração, o ID do servidor é utilizado para registrar tarefas executadas pelo robô, já o endereço IP e credenciais de acesso da IPMI são utilizados em um possível ligamento ou desligamento.

O procedimento recebe como parâmetros: ***(i)*** endereço IP do servidor, ***(ii)*** ID da primeira VM hospedado no servidor (FFD), ***(iii)*** ID do servidor no banco de dados, ***(iv)*** endereço IP da interface IPMI do servidor, ***(v)*** credenciais de acesso da IPMI, ***(vi)*** utilização do *hardware* e ***(vii)*** estado do servidor (linha 1). Além dos parâmetros recebidos, o algoritmo utiliza as variáveis limite inferior, limite de segurança e limite superior que são variáveis globais do robô.

Assim, se a pontuação do servidor for maior que o limite superior, ele será considerado sobrecarregado (linha 2). Dessa forma, o vetor *Sobrecarregados* recebe na posição ***i*** o endereço IP do servidor (linha 3), na posição ***i***+1 o ID da primeira VM hospedada nele (linha 4) e na posição ***i***+2 o ID do servidor (linha 5). Na sequência a variável ***i*** é incrementada em três

unidades para organizar, de forma correta, os dados de um outro servidor que venha a ser classificado como sobrecarregado (linha 6).

Caso a utilização do *hardware* do servidor seja maior que o limite de segurança e menor ou igual ao limite superior, o servidor é considerado quase sobrecarregado (linha 8). Dessa forma, o vetor *QuaseSobrecarregados* recebe na posição ***j*** o endereço IP do servidor (linha 9), na posição ***j***+1 o ID da primeira VM hospedada nele (linha 10), na posição ***j***+2 o ID do servidor (linha 11) e na sequência a variável ***j*** é incrementada em três unidades (linha 12).

Se a utilização do *hardware* do servidor for maior que o limite inferior e menor ou igual ao limite de segurança, o servidor é considerado normal (linha 14). Dessa forma, o vetor *Normais* recebe na posição ***k*** o endereço IP do servidor (linha 15), na posição ***k***+1 o ID da primeira VM hospedada nele (linha 16), na posição ***k***+2 o ID do servidor (linha 17) e na sequência a variável ***k*** é incrementada em três unidades (linha 18).

Se a utilização do *hardware* do servidor for menor que o limite inferior e existir VM hospedada, o servidor é considerado quase ocioso (linha 20). Dessa forma, o vetor *QuaseOciosos* recebe na posição ***l*** o endereço IP do servidor (linha 21), na posição ***l***+1 o ID da primeira VM hospedada nele (linha 22), na posição ***l***+2 o ID do servidor (linha 23) e na sequência a variável ***l*** é incrementada em três unidades (linha 24).

Se a utilização do *hardware* do servidor for menor que o limite inferior e não existir VM hospedada nele, o servidor é considerado ocioso (linha 26). Dessa forma, o vetor *Ociosos* recebe na posição ***m*** o endereço IP do servidor (linha 27), na posição ***m***+1 a *string* vazio, pois não há VM hospedada (linha 28), na posição ***m***+2 o ID do servidor (linha 29), na posição ***m***+3 o endereço IP da IPMI (linha 30), na posição ***m***+4 as credenciais de acesso da IPMI (linha 31) e na sequência a variável ***m*** é incrementada em cinco unidades (linha 32).

Se o estado do servidor for desligado, o servidor é considerado desligado (linha 34). Dessa forma, o vetor *Desligados* recebe na posição ***n*** o endereço IP do servidor (linha 35), na posição ***n***+1 a *string* vazio, pois não há VM hospedada (linha 36), na posição ***n***+2 o ID do servidor (linha 37), na posição ***n***+3 o endereço IP da IPMI (linha 38), na posição ***n***+4 as credenciais de acesso da IPMI (linha 39) e na sequência a variável ***n*** é incrementada em cinco unidades para organizar, de forma correta, os dados de um outro servidor que venha a ser classificado como desligado (linha 40).

Algoritmo 6 Wall-E - Procedimento Classificar Servidores

```
1  procedimento CLASSIFICAR(IP, ID.VM, ID.SRV, IP.IPMI, CredenciaisIPMI, UtilizacaoHW, Estado)
2      se (UtilizacaoHW > LimiteSuperior) então
3          Sobrecarregados[i] ← IP;
4          Sobrecarregados[i+1] ← ID.VM;
5          Sobrecarregados[i+2] ← ID.SRV;
6          i ← i+3;
7      fim se
8      se (UtilizacaoHW > LimiteSeguranca e UtilizacaoHW <= LimiteSuperior) então
9          QuaseSobrecarregados[j] ← IP;
10         QuaseSobrecarregados[j+1] ← ID.VM;
11         QuaseSobrecarregados[j+2] ← ID.SRV;
12         j ← j+3;
13     fim se
14     se (UtilizacaoHW > LimiteInferior e UtilizacaoHW <= LimiteSeguranca) então
15         Normais[k] ← IP;
16         Normais[k+1] ← ID.VM;
17         Normais[k+2] ← ID.SRV;
18         k ← k+3;
19     fim se
20     se (UtilizacaoHW <= LimiteInferior e ID.VM <> "vazio") então
21         QuaseOciosos[l] ← IP;
22         QuaseOciosos[l+1] ← ID.VM;
23         QuaseOciosos[l+2] ← ID.SRV;
24         l ← l+3;
25     fim se
26     se (UtilizacaoHW <= LimiteInferior e ID.VM = "vazio"e Estado = "Ligado") então
27         Ociosos[m] ← IP;
28         Ociosos[m+1] ← vazio;
29         Ociosos[m+2] ← ID.SRV;
30         Ociosos[m+3] ← IP.IPMI;
31         Ociosos[m+4] ← CredenciaisIPMI;
32         m ← m+5;
33     fim se
34     se (Estado = "Desligado") então
35         Desligados[n] ← IP;
36         Desligados[n+1] ← vazio;
37         Desligados[n+2] ← ID.SRV;
38         Desligados[n+3] ← IP.IPMI;
39         Desligados[n+4] ← CredenciaisIPMI;
40         n ← n+5;
41     fim se
42 fim procedimento
```

4.6.2 Registro das Principais Tarefas Executadas

O procedimento chamado *RegistrarTarefas* é responsável por salvar no banco de dados as principais tarefas realizadas pelo Wall-E (linha 1). Uma das motivações que levou a arma-

zenar estes registros em banco de dados foi a possibilidade de fácil visualização do número de migrações. As principais tarefas salvas no banco de dados foram: ***(i)*** ligar servidores, ***(ii)*** desligar servidores, ***(iii)*** migrar VMs entre os servidores e ***(iv)*** avisar sobre nuvem sobrecarregada.

Esse procedimento é descrito pelo Algoritmo 7, ele recebe como parâmetros: ***(i)*** usuário do banco de dados, ***(ii)*** senha do usuário do banco de dados e ***(iii)*** dados, que é uma *string* estruturada da seguinte forma: (data do evento, ação realizada, descrição da ação, ID do servidor). Após receber os parâmetros, o procedimento conecta e insere os dados no banco de dados do arcabouço, utilizando as credenciais de acesso (linha 2 e 3).

Algoritmo 7 Wall-E - Procedimento Registrar Tarefas

```
1 procedimento RegistrarTarefas(Usuario, Senha, Dados)
2 |   ConectaBancoDados(Usuario, Senha);
3 |   TabelaRegistroTarefas ← (Dados)
4 fim procedimento
```

4.6.3 Atualizar o Estado dos Servidores

O procedimento denominado *AtualizarEstado*, é responsável por atualizar o estado dos servidores no banco de dados. Os possíveis estados são: ***(i)*** ligado ou ***(ii)*** desligado. Esse procedimento era chamado toda vez que um servidor é ligado ou desligado, ele foi essencial para o funcionamento do Wall-E. Sem esse procedimento seria impossível controlar quais servidores estão ligados e quais estão desligados.

O procedimento é descrito pelo Algoritmo 8. Ele recebe como parâmetros: ***(i)*** usuário do banco de dados, ***(ii)*** senha do usuário do banco de dados, ***(iii)*** ID do servidor e ***(iv)*** o novo estado do servidor (linha 1). Na sequência do recebimento dos parâmetros, o procedimento conecta no banco de dados utilizando as credenciais (linha 2). Após a conexão percorre-se os registros da tabela, que armazena os dados dos servidores, até que se encontre a identificação única do servidor. Dessa forma, o estado do servidor é atualizado (linha 3).

Algoritmo 8 Wall-E - Procedimento Atualizar Estado dos Servidores

```
1 procedimento AtualizarEstado(Usuario, Senha, ID, NovoEstado)
2 |   ConectaBancoDados(Usuario, Senha);
3 |   TabelaServidores ← (NovoEstado, onde (ID.servidor = ID))
4 fim procedimento
```

4.6.4 Ligar e Desligar Servidores

O procedimento *LigarDesligar* é responsável por ligar e desligar servidores de computação. O ligamento ocorre quando a nuvem está sobrecarregada e o desligamento quando há servidor ocioso. O procedimento é descrito pelo Algoritmo 9. Ele recebe como parâmetros: ***(i)*** endereço IP da interface IPMI, ***(ii)*** as credenciais de acesso à IPMI, ***(iii)*** ação que será realizada e ***(iv)*** o ID do servidor (linha 1). Após receber os parâmetros, o procedimento verifica se o pedido é para ligar ou desligar o servidor (linhas 2 e 8).

Se a ação for para ligar (linha 2), é enviado o comando correspondente, os parâmetros do comando são: ***(i)*** endereço IP da interface IPMI e as ***(ii)*** credenciais de acesso à IPMI (linha 3). Após o envio do comando, é armazenado em variável a data e hora do evento (linha 4). Na sequência é estruturada a *string* denominada Dados da seguinte forma: (data do evento, ação realizada, descrição da ação) (linha 5). Com a *string* estruturada, é enviado para o procedimento *RegistrarTarefas* o usuário do banco de dados, senha do usuário do banco de dados e a *string* Dados (linha 6). Como o servidor foi ligado, há necessidade de atualizar o estado desse servidor no banco de dados. Dessa forma, é enviado para o procedimento *AtualizarEstado* o usuário do banco de dados, senha do usuário do banco de dados, o ID do servidor e o novo estado (linha 7).

Se a ação for para desligar (linha 8), faz-se necessário verificar duas condições: ***(i)*** se há pelo menos um servidor classificado como quase sobrecarregado, ou um servidor classificado como normal, ou um servidor classificado como quase ocioso e ***(ii)*** se há pelo menos dois servidores ociosos (linha 9). Se a primeira ou a segunda condição for verdadeira, o servidor poderá ser desligado. Assim, garante-se que sempre haverá pelo menos um servidor de computação ligado pronto para receber carga de trabalho, o que é algo desejável.

Se a primeira ou segunda condição for verdadeira, é enviado o comando para desligar o servidor, os parâmetros do comando são: ***(i)*** endereço IP da interface IPMI e as ***(ii)*** credenciais de acesso à IPMI (linha 10). Assim como na ação de ligar, na ação de desligar também é armazenado em variável a data e hora do evento (linha 11). Na sequência é estruturada a *string* denominada Dados da seguinte forma: (data do evento, ação realizada, descrição da ação, identificação do servidor) (linha 12). Com a *string* estruturada, é enviado para o procedimento *RegistrarTarefas* o usuário do banco de dados, senha do usuário do banco de dados e a *string* Dados (linha 13). Como o servidor foi desligado, há necessidade de atualizar o estado desse servidor no banco de dados. Dessa forma, é enviado para o procedimento *AtualizarEstado* o

usuário do banco de dados, senha do usuário do banco de dados, o ID do servidor e o novo estado (linha 14).

Algoritmo 9 Wall-E - Procedimento Ligar Desligar Servidores

```
1  procedimento LigarDesligar(IP_IPMI, CredenciaisIPMI, Acao, ID.servidor)
2      se (Acao = Ligar) então
3          ComandoLigar → (IP_IPMI, CredenciaisIPMI);
4          DataEvento ← DataServidor;
5          Dados ← (DataEvento, Ligar, "Foi ligado.", ID.servidor);
6          RegistrarTarefas(UsuarioBD, SenhaBD, Dados);
7          AtualizarEstado(UsuarioBD, SenhaBD, ID.servidor, Ligado);
8      senão se (Acao = Desligar) então
9          se [(QuaseSobrecarregados ou Normais ou QuaseOciosos > 0) ou (Ociosos > 1)] então
10             ComandoDesligar → (IP_IPMI, CredenciaisIPMI);
11             DataEvento ← DataServidor;
12             Dados ← (DataEvento, Desligar, "Foi desligado.", ID.servidor);
13             RegistrarTarefas(UsuarioBD, SenhaBD, Dados);
14             AtualizarEstado(UsuarioBD, SenhaBD, Desligado, ID.servidor);
15         fim se
16     fim se
17 fim procedimento
```

4.6.5 Migrar VMs entre os Servidores

Esse procedimento é responsável por enviar a solicitação de migração de VM para o servidor controlador da nuvem computacional. Ele é descrito pelo Algoritmo 10. Ele recebe como parâmetros: ***(i)*** ID da VM ***(ii)*** endereço IP do servidor de destino e ***(iii)*** ID do servidor (linha 1). Após receber os parâmetros é requisitado, ao servidor controlador, um *token* de autenticação. Para isso há necessidade de utilizar credenciais de acesso da nuvem (usuário e senha) (linha 2). Com o *token*, o ID da VM que deseja migrar e o endereço IP do servidor de destino, é enviado uma requisição de migração, ao servidor controlador (linha 3).

Para registrar a tarefa executada, é armazenado em variável a data e hora da migração (linha 4). Na sequência é estruturada a *string* denominada Dados da seguinte forma: (data do evento, ação realizada, descrição da ação, identificação do servidor) (linha 5). Com a *string* estruturada, é enviado para o procedimento *RegistrarTarefas* o usuário do banco de dados, senha do usuário do banco de dados e a *string* Dados (linha 6).

Algoritmo 10 Wall-E - Procedimento Migrar VM

```
1  procedimento Migrar(ID_VM, EnderecoIPdestino, ID.servidor)
2      Token ← [RequisitaAutenticacao → URL_Autenticacao(UsuarioNuvem, Senha)];
3      RequisitaMigracao → URL_API(Token, controlador(ID_VM, EnderecoIPdestino));
4      DataEvento ← DataServidor;
5      Dados ← (DataEvento, Migrar, ID.servidor, "Recebeu a VM de ID : ID_VM.");
6      Registrar_Tarefas(UsuarioBD, SenhaBD, Dados);
7  fim procedimento
```

4.6.6 Decisões de Migração de VMs, Ligamento e Desligamento de Servidores

Esse procedimento é responsável por decidir sobre migrações de VMs, ligamento ou desligamento de servidores de computação. Desse modo, ele é considerado o cérebro do Wall-E. Em cada execução do algoritmo, é tomada no máximo uma decisão. Isso visa, por exemplo, não sobrecarregar os servidores com migrações simultâneas. Apesar da principal motivação do trabalho ser a economia de energia elétrica em *data centers*, o algoritmo tenta não interferir de forma negativa no desempenho da nuvem. Primeiramente, caso haja servidor sobrecarregado, ele balanceia a carga entre os servidores de computação. Somente quando não houver servidor classificado como sobrecarregado é que o algoritmo buscará a consolidação de VMs e, posteriormente, o desligamento de servidores.

O procedimento *Decidir* é descrito pelo Algoritmo 11. O algoritmo utiliza os vetores de classificação do procedimento *Classificar*. Esses vetores são variáveis globais do robô. Assim, após ser chamado (linha 35), ele verifica se o tamanho do vetor *Sobrecarregados* é maior que zero (linha 2). Em caso verdadeiro, significa que existe servidor sobrecarregado. Assim, é armazenado em variável o ID da primeira VM do primeiro servidor sobrecarregado (linha 3). Na sequência, o algoritmo verifica se há servidor com capacidade de receber a VM do servidor sobrecarregado (linha 4 a 11). As prioridades de migração com base no servidor de destino são as seguintes:

- **Prioridade 0**: migrar para um servidor classificado como normal;
- **Prioridade 1**: migrar para um servidor classificado como quase ocioso;
- **Prioridade 2**: migrar para um servidor classificado como ocioso e
- **Prioridade 3**: ligar um servidor, caso exista, para posterior migração.

Assim, o algoritmo verifica primeiro se o tamanho do vetor *Normais* é maior que zero. Em caso verdadeiro, significa que existe servidor classificado como normal (linha 4). Dessa forma, para realizar a migração, os seguintes parâmetros são enviados para o procedimento *Migrar*: **(*i*)** ID da VM, **(*ii*)** o endereço IP do servidor destino, que está na posição zero do vetor, e **(*iii*)** o ID do servidor de destino, que está na posição 2 do vetor (linha 5).

Se não existir servidor classificado como normal, o algoritmo verifica se o tamanho do vetor *QuaseOciosos* é maior que zero (linha 6). Em caso verdadeiro, significa que existe servidor classificado como quase ocioso. Dessa forma, para realizar a migração, os seguintes parâmetros são enviados para o procedimento *Migrar*: **(*i*)** ID da VM, **(*ii*)** o endereço IP do servidor de destino e **(*iii*)** o ID do servidor de destino (linha 7).

Se não existir servidor classificado como normal e quase ocioso, o algoritmo verifica se o tamanho do vetor *Ociosos* é maior que zero (linha 8) Em caso verdadeiro, significa que existe servidor classificado como ocioso. Dessa forma, para realizar a migração, os seguintes parâmetros são enviados para o procedimento *Migrar*: **(*i*)** ID da VM, **(*ii*)** o endereço IP do servidor de destino e **(*iii*)** o ID do servidor de destino (linha 9).

Se não existir servidor classificado como normal, quase ocioso e ocioso, o algoritmo verifica se o tamanho do vetor *Desligados* é maior que zero (linha 10). Em caso verdadeiro, significa que existe servidor desligado. Dessa forma, é necessário realizar o ligamento do primeiro servidor classificado como desligado para posteriormente receber VMs. Para realizar o ligamento, os seguintes parâmetros são enviados para o procedimento *LigarDesligar*: **(*i*)** o endereço IP da IPMI do primeiro servidor desligado, que está na posição 3 do vetor, **(*ii*)** as credenciais de acesso da IPMI, que está na posição 4 do vetor, **(*iii*)** a ação que será realizada, que no caso é ligar e **(*iv*)** o ID do servidor a ser ligado, que está na posição 2 do vetor (linha 11).

Caso exista servidor classificado como sobrecarregado e não exista servidor classificado como normal ou quase ocioso ou ocioso ou desligado, significa que a nuvem está sobrecarregada. Dessa forma, a única ação a ser realizada é registrar no banco de dados a possível degradação de desempenho (linhas 12 a 17).

Se não existir servidor classificado como sobrecarregado, é verificado se o tamanho do vetor *Ociosos* é maior que zero (linha 18). Em caso verdadeiro, significa que existe servidor classificado como ocioso. Assim, é enviada uma requisição de desligamento (linha 19). Para realizar o desligamento, do primeiro servidor classificado como ocioso, os seguintes parâmetros são enviados para o procedimento *LigarDesligar*: **(*i*)** o endereço IP da IPMI do primeiro servidor ocioso, que está na posição 3 do vetor, **(*ii*)** as credenciais de acesso da IPMI, que está na posição 4 do vetor, **(*iii*)** a ação que será realizada, que no caso é desligar e **(*iv*)** o ID do servidor a ser desligado, que está na posição 2 do vetor.

Em caso de não existência de servidor sobrecarregado ou servidor ocioso, é verificado se existe servidor quase ocioso. Assim, compara-se se o tamanho do vetor *QuaseOcioso* é maior que zero (linha 20). Se for maior, significa que existe servidor classificado como quase ocioso. Dessa forma, faz-se necessário verificar se existe servidor com capacidade para receber as VMs hospedadas neste servidor quase ocioso (linhas 22 a 26). O intuito é fazer com que este servidor se transforme em ocioso, sem sobrecarregar outros servidores, para posterior desligamento.

Desse modo, é armazenado em variável, o ID da primeira VM do primeiro servidor quase ocioso (linha 21).

Na sequência, o algoritmo verifica se o tamanho do vetor *Normais* é maior que zero. Em caso verdadeiro, significa que existe servidor classificado como normal. Dessa forma, pode-se realizar a migração da VM do servidor quase ocioso para o primeiro servidor classificado como normal. Para realizar a migração, os seguintes parâmetros são enviados para o procedimento *Migrar*: **(*i*)** ID da VM, **(*ii*)** o endereço IP do servidor destino, que está na posição zero do vetor, e **(*iii*)** o ID do servidor de destino, que está na posição 2 do vetor (linha 23).

Se não existir servidor classificado como normal, é verificado se existem pelo menos dois servidores classificados como quase ociosos. A intensão é consolidar as VMs, por exemplo, de dois servidores classificados como quase ociosos em apenas um e desligar o outro. Dessa forma, é comparado se o tamanho do vetor *QuaseOciosos* é maior que 3 (linha 24). Em caso verdadeiro, significa que existe pelo menos dois servidores classificados como quase ociosos. Assim, pode-se realizar a migração da VM do primeiro servidor quase ocioso para o segundo servidor classificado como quase ocioso. Para realizar a migração, os seguintes parâmetros são enviados para o procedimento *Migrar*: o **(*i*)** ID da VM, **(*ii*)** o endereço IP do servidor destino, que está na posição 3 do vetor, e **(*iii*)** o ID do servidor de destino, que está na posição cinco do vetor (linha 25).

Na sequência, são declaradas as variáveis globais do Wall-E. Primeiramente, são declarados os limites que são utilizados para decidir a classificação de cada servidor. Os limites são valores informados, por especialista, por meio do kWh-Web. Assim, esses valores são pesquisados no banco de dados e inseridos no Wall-E (linha 29). Em seguida, são declarados os vetores (linha 30), que possibilitam organizar os servidores de acordo com suas classificações. Por fim, são declaradas as variáveis de controle de posições dos vetores (linha 31). Tanto os vetores como as variáveis de controle de posição, necessitam ser variáveis globais. Isso se deve ao fato de que são utilizadas por mais de um procedimento do Wall-E.

Para que aconteçam as migrações de VMs, ligamento e desligamento de servidores, os procedimentos *Classificar* e *Decidir* precisam ser chamados. Para chamar o procedimento *Classificar*, existe um laço de repetição que envia os parâmetros necessários. A repetição acontece de acordo com a quantidade de servidores (linhas 32 a 34). Primeiro, os servidores são classificados, na sequência o robô decide se haverá necessidade de realizar alguma ação (linha 35).

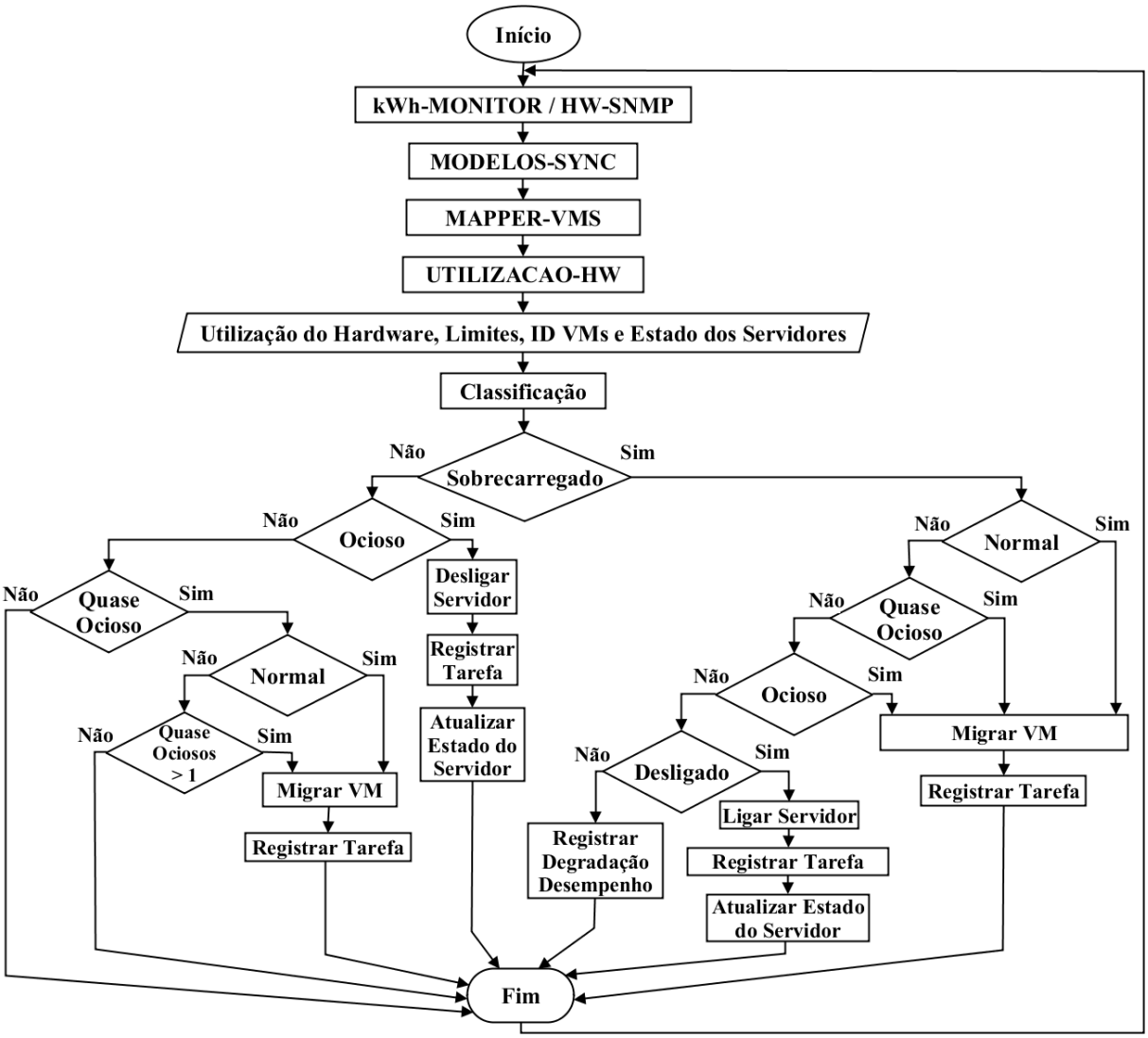
Algoritmo 11 Wall-E - Tomada de Decisões

```
1  procedimento DECIDIR()
2      se (Tamanho Vetor Sobrecarregados > 0) então
3          VMSobrecarregado ← Sobrecarregados[1];
4          se (Tamanho Vetor Normais > 0) então
5              Migrar(VMSobrecarregado, Normais[0], Normais[2]);
6          senão se (Tamanho Vetor QuaseOciosos > 0) então
7              Migrar(VMSobrecarregado, QuaseOcioso[0], QuaseOciosos[2]);
8          senão se (Tamanho Vetor Ociosos > 0) então
9              Migrar(VMSobrecarregado, Ociosos[0], Ociosos[2]);
10         senão se (Tamanho Vetor Desligados > 0) então
11             LigarDesligar(Desligados[3], Desligados[4], Ligar, Desligados[2]);
12         senão
13             DataEvento ← DataServidor;
14             Descricao("Esta sobrecarregado e nao ha no disponivel.");
15             Dados ← (DataEvendo, "Aviso", Descricao, Sobrecarregados[2]);
16             RegistrarTarefas(Dados, UsuarioBD, SenhaBD);
17         fim se
18     senão se (Tamanho Vetor Ociosos > 0) então
19         LigarDesligar(Ociosos[3], Ociosos[4], Desligar, Ociosos[2]);
20     senão se (QuaseOciosos > 0) então
21         VMQuaseOcioso ← QuaseOciosos[1];
22         se (Tamanho Vetor Normais > 0) então
23             Migrar(VMQuaseOcioso, Normais[0], Normais[2]);
24         senão se (Tamanho Vetor QuaseOciosos > 3) então
25             Migrar(VMQuaseOcioso, Ociosos[3], Ociosos[5]);
26         fim se
27     fim se
28 fim procedimento
29 LimiteSuperior; LimiteSeguranca; LimiteInferior ← Banco de dados;
30 Sobrecarregados[]; QuaseSobrecarregados[]; QuaseOciosos[]; Ociosos[]; Desligados[];
31 i = 0; j = 0; k = 0; l = 0; m = 0; n = 0;
32 para (b ← 1; b <= NumeroServidores; b++) faça
33     CLASSIFICAR(IP.b, ID.VM.b, UtilizacaoHW.b, Estado.b, ID.SRV.b, IP.IPMI.b, CredIPMI.b)
34 fim para
35 Chamada → Decidir
```

A Figura 16 mostra, de forma abstrata, todo fluxo de execução dos *middleware*. Todos eles são executados pelo *daemon* de agendamento de tarefas do sistema operacional. Os primeiros a serem executados são o kWh-MONITOR e HW-SNMP, que são os *middleware* de monitoramento. Na sequência executa-se o MODELOS-SYNC, que sincroniza os modelos de VMs criados na plataforma de nuvem com o banco de dados do arcabouço. Logo após, é executado o MAPPER-VMs, que mapeia a localização de cada uma das VMs. Após o mapeamento das VMs, é executado o UTILIZACAO-HW, que calcula a utilização do *hardware* de cada um dos servidores de computação que estão ligados. A classificação dos servidores é realizada com

base na saída do UTILIZACAO-HW, mais os limites, IDs das VMs e o estado de cada um dos servidores. E é com base nesta classificação que ocorrem as ações sobre a nuvem.

Figura 16 – Fluxograma de Execução do Arcabouço

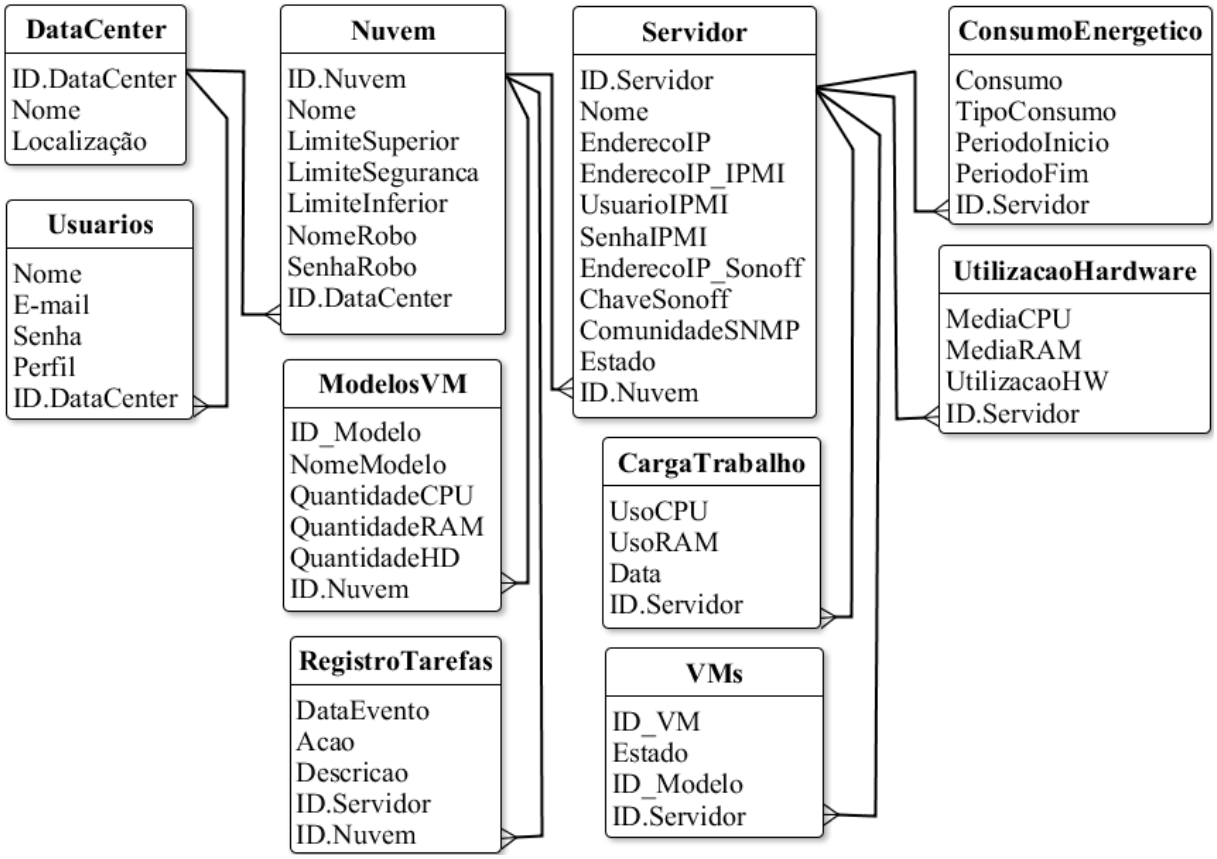


4.6.7 kWh-Web

O arcabouço utiliza um banco de dados para armazenar os dados sobre o ***(i)*** consumo energético, a ***(ii)*** utilização do *hardware*, a ***(iii)*** nuvem, os ***(iv)*** servidores, as ***(v)*** VMs, entre outros. Para facilitar o gerenciamento deste banco de dados, desenvolveu-se uma aplicação *Web* denominada kWh-Web. Além de facilitar o gerenciamento do banco de dados, a kWh-Web é responsável por gerar os parâmetros dos *middleware* e do robô Wall-E. Como exemplos de parâmetros, pode-se citar o ID da primeira VM hospedada em um determinado servidor, o estado do servidor (ligado ou desligado), a comunidade SNMP e usuário e senha que o robô utiliza para autenticar na nuvem.

A Figura 17 mostra a estrutura de tabelas do banco de dados do arcabouço, os principais campos de cada tabela e o relacionamento entre as tabelas. Os campos das tabelas, *DataCenter*, *Usuarios*, *Nuvem* e *Servidor* são cadastrados pelo administrador do *data center*. As tabelas *ConsumoEnergetico* e *CargaTrabalho* são utilizadas para armazenar os dados do monitoramento, a tabela *UtilizacaoHardware* armazena a utilização do *hardware* de cada um dos servidores de computação, já a tabela *VMs* armazena a localização de cada uma das VMs, a tabela *ModelosVM* armazena as informações referentes aos modelos de VMs criados através da plataforma de nuvem e a tabela *RegistroTarefas* armazena as tarefas realizadas pelo Wall-E.

Figura 17 – Estrutura das Tabelas do Banco de Dados do Arcabouço



5 MATERIAIS E MÉTODOS

Nesta seção, são apresentados os materiais é métodos utilizados nos experimentos. O principal objetivo destes experimentos foi validar o arcabouço e quantificar a economia energética que ele propicia. Além disso, outro objetivo foi analisar o impacto causado pelo arcabouço no desempenho da nuvem computacional. Na próxima subseção, é possível observar as principais configurações dos servidores, do *switch*, do roteador *wireless* e dos wattímetros. Além disso, também é possível observar uma breve descrição e a versão de cada um dos *software*. Já na Subseção 5.2 são apresentados os métodos utilizados na montagem do servidor NFS, da nuvem computacional, do servidor que hospedou o arcabouço, dos experimentos iniciais e experimentos de validação do arcabouço.

5.1 Materiais

Com exceção dos wattímetros e do roteador *wireless*, todos os demais equipamentos necessários para realização do trabalho, foram disponibilizados pela UFLA. Todos os *hardware* ficaram alocados no *data center* de pesquisas do Departamento de Ciência da Computação - GrubiCom. Já os *software*, foram baixados da Internet e todos possuíam licença livre ou *open source*.

5.1.1 Servidores

Para a formação do ambiente de testes, foram utilizados seis servidores. Quatro desses servidores foram utilizados na nuvem computacional. Um servidor utilizado para hospedar o arcabouço e gerar carga de trabalho nas VMs e um servidor foi utilizado como NFS (*Network File System*), que era responsável por armazenar os vHDs das VMs. As principais configurações dos servidores, função e quantidade, estão descritas na Tabela 2.

Tabela 2 – Servidores Utilizados

Servidores	Quantia	Marca	Modelo	CPU	RAM	HD
Servidores da Nuvem	1 controlador 3 computação	Dell	PowerEdge R620	Intel Xeon E5-2609 Quad Core 2,5GHz	2 pentes DDR3 de 16 GB e 1600 MHz	4 HDs SAS de 320 GB
Hospedagem Arcabouço e Geração de Carga de Trabalho	1	IBM	System x3400 M3	Intel E5-640 Quad Core 2,6GHz	4 pentes DIMM de 4GB e 1067 MHz	2 HDs de 1TB
NFS	1	HP	Compaq 6005	Intel Core 2 Quad Q8400 2.66GHz	2 pentes DDR3 de 2 GB e 1333 MHz	1 HD de 500 GB

5.1.2 Switch, Roteador Wireless e Wattímetros

Para construir a rede de computadores, utilizou-se um *switch* e um roteador *wireless*. O *switch* era responsável por possibilitar a comunicação entre todos servidores e entre o servidor que hospedou o arcabouço e a IPMI (iDrac6). Já o roteador *wireless* possibilitou a comunicação entre o servidor que hospedou o arcabouço e os wattímetros Sonoff Pow. Os wattímetros tinham a função de possibilitar o monitoramento do consumo energético dos servidores da nuvem computacional. As principais características desses equipamentos estão descritas na Tabela 3.

Tabela 3 – Principais Características do Switch, Roteador Wireless e Wattímetros

Equipamento	**Quantia**	**Principais Características**	**Marca**	**Modelo**
Switch	**1**	24 portas de 10/100 Mbps	Extreme	X150-24t
Roteador Wireless	**1**	Compatível com os padrões 802.11bg, frequência 2,4 a 2.4835 GHz	TP-link	TL-WR741ND
Wattímetros	4	Interface wireless, exatidão de ±1% e firmware ESPurna versão 1.13.0	Itead	Sonoff Pow
	4	Interface Ethernet, exatidão de ±7%, vem embarcado nos servidores.	Dell	IPMI - iDrac6

A Figura 18 mostra de um lado os servidores da nuvem, o *switch*, o servidor que hospedou o arcabouço e o servidor NFS. Do outro lado, mostra os wattímetros Sonoff Pow monitorando o consumo energético dos servidores.

Figura 18 – Servidores, Switch e Wattímetros no Data Center GrubiCom

5.1.3 Software

Foram empregados *software* livres e *open source*. Os sistemas operacionais Ubuntu e Debian foram baixados dos sites oficiais, assim como outros pacotes. A Tabela 4 mostra os *software* utilizados no trabalho, a versão e uma breve descrição.

Tabela 4 – Software Utilizados

Software	Versão	Descrição
Ubuntu	Server 16.04 de 64 *bits*	Distribuição do sistema operacional Linux amplamente utilizada para uso pessoal e que vem sendo utilizada em servidores (CANONICAL, 2018a).
Debian	9.2 de 64 *bits*	Distribuição do sistema operacional Linux amplamente utilizada em servidores (DEBIAN, 2018).
OpenStack	Ocata	15º versão do OpenStack, teve a capacidade de gerenciamento, escalabilidade e resiliência melhorada (OPENSTACK, 2017c).
IPMItool	1.8.16	Fornece uma interface de linha de comando, que possibilita gerenciar as funções da IPMI. Dentre estas funções está a leitura de sensores. No gerenciamento remoto o protocolo de transporte utilizado é o UDP (INTEL et al., 2013).
cURL	7.47.0	Ferramenta de linha de comando utilizada para transferir dados com URLs (*Uniform Resource Locator*). Dentre os vários protocolos suportados, pode-se citar o HTTP e HTTPs (STENBERG; FANDRICH, 2018).
Apache	2.4	Servidor HTTP (*Web*) amplamente utilizado (APACHE, 2018c).
PHP	7.0.15	*Hypertext Preprocessor* é uma linguagem de programação interpretada, o que a torna interessante para o desenvolvimento *Web* (PHPGROUP, 2017).
MySQL	5.7.18	Sistema gerenciador de banco de dados relacional, que utiliza linguagem SQL (ORACLE, 2017).
JMeter	2.11.20151206	Aplicação gráfica, 100% feita em Java desenvolvida para executar testes funcionais e medir o desempenho de aplicações Web (APACHE, 2018a).
ESPurna	1.13.0	*Firmware* personalizado para interruptores *smart* baseados no *chip* ESP8266. Ele utiliza a estrutura do Arduino Core e várias bibliotecas de terceiros. Dentre as funcionalidades, pode-se destacar a possibilidade de monitorar o consumo energético por meio de uma API REST (PEREZ, 2018).
Stress-ng	0.05.23-1	Ferramenta para gerar carga de trabalho e estressar um sistema computacional com sistemas Linux (CANONICAL, 2018b).

5.2 Métodos

Nesta subseção, apresentam-se os métodos utilizados, pode-se observar a metodologia de montagem do ambiente de experimentos, os tipos e intensidades de cargas de trabalhos que as VMs receberam e os experimentos realizados.

5.2.1 Implementação do Servidor NFS

Para a instalação e configuração do NFS, foi utilizado o manual oficial (CANONICAL, 2018c). Este servidor foi utilizado para armazenar os vHDs das VMs. Essa centralização agiliza o processo de migração de VMs. Isso porque, em vez de copiar todo vHD da VM, de um servidor de computação para outro, copia-se somente o conteúdo da vRAM.

5.2.2 Implementação da Nuvem OpenStack

Para a implementação da nuvem computacional, instalou-se em todos os servidores da nuvem, o sistema operacional Ubuntu. Esse sistema operacional é um dos recomendados para instalação do OpenStack. Após a instalação do sistema operacional, implementou-se a nuvem computacional com a versão Ocata da plataforma, utilizando-se o tutorial oficial (OPENSTACK, 2018c). Um servidor foi configurado como controlador e três como servidor de computação. Além disso, configurou-se a migração a quente utilizando a abordagem pré-cópia com armazenamento compartilhado (NFS), também de acordo com a documentação oficial (OPENSTACK, 2019).

Após instalação e configuração da nuvem, instanciou-se 30 VMs. Como o OpenStack utiliza um algoritmo *round-robin* para distribuir as VMs entre os servidores de computação, inicialmente cada um dos 3 servidores hospedou 10 VMs (OPENSTACK, 2018b). Na busca de reproduzir um ambiente mais próximo da realidade, foram criados 4 modelos diferentes de VMs. Os modelos utilizados e a quantidade de VMs criadas com cada modelo são apresentados na Tabela 5.

Tabela 5 – Modelos e Quantidade de VMs

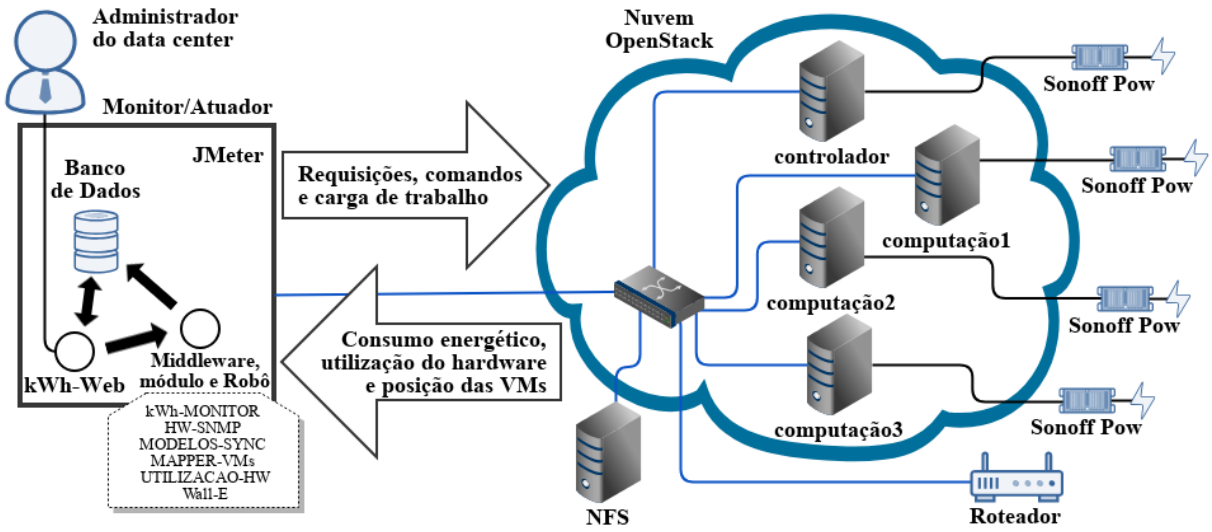
Modelo	**vCPUs**	**vRAM**	**vHD**	**Quantidade de VMs**
A-Grande	1	4GB	8GB	3
B-Médio	1	3GB	4GB	7
C-Pequeno	1	1GB	2GB	10
D-Mínimo	1	512MB	2GB	10

O sistema operacional escolhido para as VMs foi o Debian. Além disso, instalou-se em cada VM, a ferramenta Stress-ng e Apache Web Server. As VMs hospedaram arquivos HTML (*HyperText Markup Language*) e imagens JPEG (*Joint Photographics Experts Group*) que somavam um total de 464 *kBytes*. Esses arquivos eram requisitados durante os experimentos e a finalidade foi avaliar a quantidade de requisições atendidas.

5.2.3 Implementação do Servidor de Hospedagem do Arcabouço

O arcabouço foi hospedado no servidor denominado Monitor/Atuador. O sistema operacional utilizado foi o Ubuntu. Além disso, instalou-se os seguintes *software* que foram necessários para que o arcabouço funcionasse: ***(i)*** Apache, ***(ii)*** MySQL ***(iii)*** PHP, ***(iv)*** cURL e ***(v)*** IPMItool. Os *middleware*, o módulo para cálculo da utilização do *hardware* e o Wall-E, foram escritos em *Shell Script* e ficaram sendo executados, a cada minuto, pelo *Crontab* do sistema operacional. Esse servidor também foi utilizado para gerar carga de trabalho nas VMs da nuvem, dessa forma, também se instalou nele o JMeter. A Figura 19 mostra a arquitetura do ambiente de experimentos, na qual é possível visualizar os servidores e os principais fluxos de comunicação entre eles.

Figura 19 – Arquitetura do Ambiente de Experimentos



5.3 Experimentos Iniciais

Antes dos experimentos de validação do arcabouço, realizou-se um total de seis experimentos. Dois desses experimentos foram utilizados para decidir qual wattímetro utilizar nos experimentos de validação do arcabouço. O objetivo foi validar a exatidão, informada pelos fabricantes, dos wattímetros da IPMI (iDrac6) (±7%) e Sonoff Pow (±1%). Na sequência, realizou-se os outros quatro experimentos. Estes experimentos foram utilizados para decidir qual método de economia de energia aplicar nos servidores da nuvem. Os métodos possíveis eram: ***(i)*** colocar os servidores no modo suspenso ou ***(ii)*** desligá-los. A metodologia destes experimentos iniciais, são apresentados na sequência.

Validação da Exatidão dos Wattímetros

Para validar a exatidão, informada pelos fabricantes dos wattímetros da IPMI (iDrac6) e Sonoff Pow, foram realizados dois experimentos. No primeiro, foi monitorado o consumo energético da nuvem computacional utilizando o wattímetro da IPMI (iDrac6I), que já vem embarcado nos servidores. Já no segundo experimento, monitorou-se o consumo energético utilizando os wattímetros Sonoff Pow. Cada experimento teve duração de 16 horas e a nuvem computacional recebeu a mesma carga de trabalho durante os dois experimentos.

Consumo Energético de Servidores Ociosos, Suspensos e Desligados

Trabalhos como de (BELOGLAZOV; BUYYA, 2012) apontam, como solução para economizar energia, colocar servidores ociosos no modo suspenso. Para validar esse método, em ambiente real, realizou-se quatro experimentos com duração de 16 horas cada um. O primeiro experimento foi utilizado para controle. Nele, os servidores ficaram ligados e recebendo cargas de CPU, RAM e HD. As cargas de CPU e RAM eram de 100% da capacidade de cada item. Já no HD, de cada um dos servidores, era escrito, lido e apagado um arquivo de 2GB de forma constante. A ferramenta utilizada para gerar estas cargas foi a Stress-ng e o comando utilizado foi: *# stress-ng -t 16h --cpu 4 --cpu-load 100 --vm 7 --vm-bytes 4096M --hdd 1 --hdd-bytes 2G*.

No segundo experimento os servidores ficaram ligados, porém sem receber carga de trabalho. No terceiro experimento os servidores foram colocados no modo suspenso. Por fim, no quarto experimento, os servidores foram desligados.

5.4 Experimentos de Validação do Arcabouço

Para verificar a eficiência do arcabouço, realizou-se um total de dezesseis experimentos. O primeiro experimento foi utilizado para controle. Nele, o robô Wall-E foi desativado. Assim, o OpenStack funcionou sem interferências e não ocorreram migrações e desligamento de servidores. Nos demais experimentos, o Wall-E foi habilitado e os limites superior, de segurança e inferior variavam. O objetivo, além de validar a eficiência do arcabouço, foi observar a relação entre os limites de utilização do *hardware*, quantidade de migrações, economia de energia, aumento da latência das mensagens HTTP, quantidade de requisições HTTP atendidas e não atendidas. A Tabela 6 apresenta a configuração dos experimentos, em que é possível observar o tipo de alocação de VMs e os limites utilizados.

Tabela 6 – Limites Durante os Experimentos

Experimento	Tipo de Alocação de VMs	Limites (%)		
		LSu[b]	LSe[c]	LI[d]
1	Estática [a]	-	-	-
2	Dinâmica	50	40	10
3	Dinâmica	50	40	20
4	Dinâmica	50	40	30
5	Dinâmica	70	60	10
6	Dinâmica	70	60	20
7	Dinâmica	70	60	30
8	Dinâmica	70	60	40
9	Dinâmica	70	60	50
10	Dinâmica	90	80	10
11	Dinâmica	90	80	20
12	Dinâmica	90	80	30
13	Dinâmica	90	80	40
14	Dinâmica	90	80	50
15	Dinâmica	90	80	60
16	Dinâmica	90	80	70

[a] Posicionamento inicial das VMs (subseção 5.2.2)

[b] Limite Superior, [c] Limite de Segurança e [d] Limite Inferior

Cada experimento teve duração de dezesseis horas. Durante todo tempo, as VMs receberam cargas de trabalho. Foram gerados dois tipos de carga: ***(i)*** requisições HTTP e ***(ii)*** cargas de CPU e RAM. Para gerar as requisições HTTP, utilizou-se o JMeter e para gerar carga na CPU e RAM utilizou-se o Stress-ng. Os comandos do Stress-ng eram enviados para as VMs, via SSH (*Secure Shell*).

Além disso, havia duas intensidades de carga: ***(i)*** carga do dia e ***(ii)*** carga da noite. A carga do dia era uma carga mais intensa e a da noite mais branda. Durante as 16 horas de cada experimento, a intensidade das cargas variava. Nas primeiras quatro horas, as VMs recebiam carga do dia, nas próximas quatro horas, recebiam carga da noite, depois mais quatro horas com carga do dia e, por fim, mais quatro horas com carga da noite. O objetivo foi avaliar o comportamento do arcabouço com mudanças bruscas de cargas. A Figura 20, mostra as intensidades das cargas de trabalho de acordo com o tempo e a Tabela 7 detalha as cargas de trabalho por tipo e intensidade.

Figura 20 – Intensidades das Cargas de Acordo com o Tempo

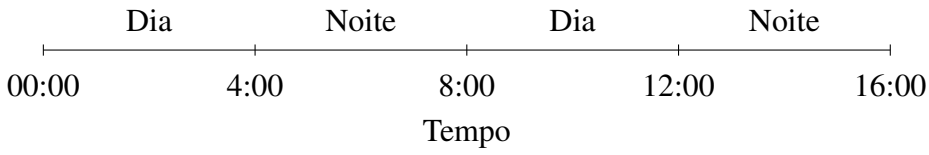


Tabela 7 – Carga de Trabalho por Tipo e Intensidade

Intensidade	HTTP (JMeter)	CPU e RAM (Stress-ng)
Dia	Requisições HTTP constantes, de todos os arquivos HTML e JPEG. Foi configurado o temporizador aleatório gaussiano, com 100 milissegundos entre uma requisição e outra, e variação de até 50 milissegundos. Maiores detalhes podem ser consultados no manual oficial (APACHE, 2018b).	Carga constante de 1% na vCPU. Três processos simultâneos de alocação e desalocação de 50MB de vRAM, com intervalos de 20 segundos entre alocação e desalocação.[a]
Noite	Requisições HTTP constantes, de todos os arquivos HTML e JPEG. Foi configurado o temporizador aleatório gaussiano, com 10.000 milissegundos entre uma requisição e outra, e variação de até 50 milissegundos. Maiores detalhes podem ser consultados no manual oficial (APACHE, 2018b).	Carga constante de 1% na vCPU. Um processo alocação e desalocação de 25MB de vRAM, com intervalos de 40 segundos entre alocação e desalocação.[b]

[a] ssh -i chave_acesso.pem debian@vms "sudo -i stress-ng -t 4h –cpu 1 –cpu=load 1 –vm 3 –vm-bytes 50M –vm-hang 20"

[b] ssh -i chave_acesso.pem debian@vms "sudo -i stress-ng -t 4h –cpu 1 –cpu=load 1 –vm 1 –vm-bytes 25M –vm-hang 40"

Nos experimentos com alocação dinâmica de VMs foi necessário estabelecer um fluxo de ações. O principal objetivo destas foi reiniciar a nuvem computacional, deixando-a idêntica ao estado inicial de implantação, conforme descrito na Subseção 5.2.2. Isso foi necessário pois, ao final de cada experimento, havia VMs fora dos servidores originais e servidores desligados. O fluxo de ações foi o seguinte:

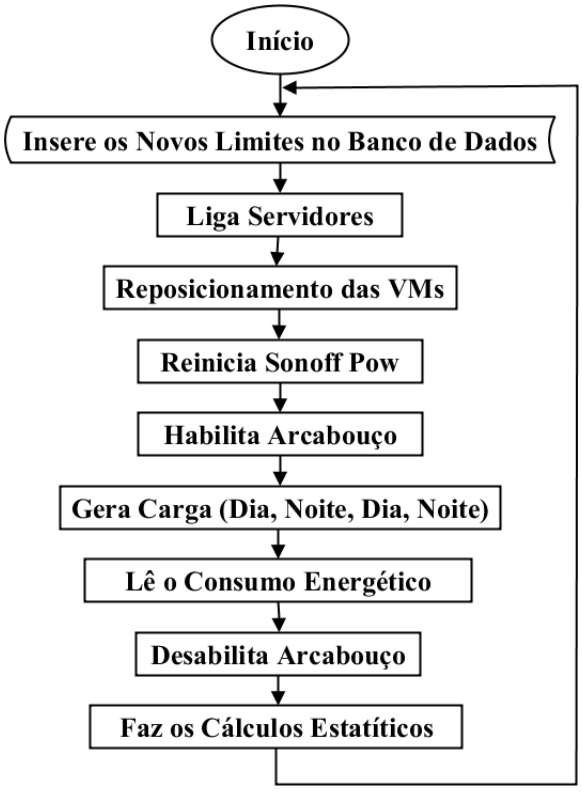
1. Inserir os novos limites no banco de dados;
2. Ligar os servidores que foram desligados no experimento anterior;
3. Reposicionar as VMs conforme posição inicial;
4. Zerar consumo energético acumulado dos wattímetros;
5. Habilitar o Robô Wall-E;
6. Gerar cargas por 16 horas;
7. Ler o consumo energético acumulado dos wattímetros;
8. Desabilitar o Robô Wall-E;
9. Fazer os cálculos estatísticos.

Das ações realizadas, duas merecem maior atenção. Foi necessário desabilitar o robô Wall-E ao final de cada um dos experimentos com alocação dinâmica de VMs, pois no momento em que fosse ligar os servidores que foram desligados no experimento anterior e reposicionar as VMS, o robô tentaria consolidar as VMs e desligar servidores. Assim, não deixaria colocar a nuvem no estado inicial. Já os cálculos estatísticos eram a soma de requisições atendidas, a

média da latência das mensagens HTTP, a quantidade de perda de mensagens HTTP, a média de utilização do *hardware* de cada um dos servidores e a quantidade de migrações realizadas. Esses parâmetros foram utilizados para realizar a análise da degradação, causada pelo arcabouço, no desempenho da nuvem computacional. A Figura 21 apresenta o fluxo de ações.

É importante ressaltar que os experimentos de validação do arcabouço têm limitações. Como exemplo, pode-se citar: ***(i)*** as cargas de trabalhos aplicadas não representam o comportamento de cargas reais, apesar de variarem durante o tempo, ***(ii)*** o número reduzido de servidores que compõem a nuvem computacional pode restringir a análise do comportamento do arcabouço e ***(iii)*** o servidor NFS, que armazenou os vHDs das VMs, não é o equipamento mais apropriado para esta finalidade e pode ter criado gargalos durante as migrações de VMs; os *storages* são mais adequados para esta finalidade.

Figura 21 – Fluxograma de Execução dos Experimentos com Alocação Dinâmica de VMs



6 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os resultados de todos os experimentos. Primeiramente, são apresentados os resultados dos experimentos iniciais. Que foram utilizados para decidir quais wattímetros utilizar nos experimentos de validação do arcabouço e qual método aplicar nos servidores da nuvem para economia energética. Por fim, são apresentados os resultados dos experimentos de validação do arcabouço.

6.1 Experimentos Iniciais

Esta subseção apresenta os resultados dos experimentos iniciais. Que tiveram como objetivos principais, fundamentar as decisões de quais wattímetros utilizar durante os experimentos de validação do arcabouço e qual método de economia de energia aplicar nos servidores da nuvem.

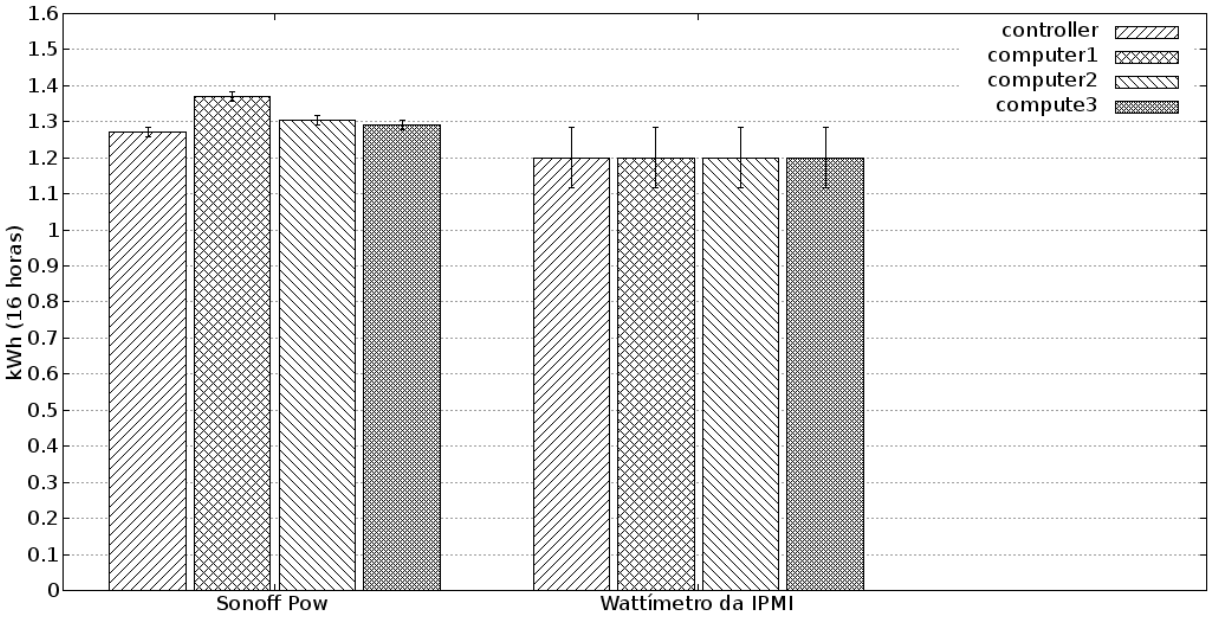
6.1.1 Validação da Exatidão dos Wattímetros

Durante o primeiro experimento, no qual utilizou-se os wattímetros Sonof Pow, o consumo foi de 5,238 kWh. No segundo experimento, onde utilizou-se os wattímetros da IPMI (iDrac6), o consumo energético total da nuvem foi de 4,800 kWh. A Figura 22, mostra o consumo energético de cada servidor, durante os dois experimentos e a exatidão dos wattímetros. Respeitando a exatidão de cada wattímetros, IPMI (±7%) e Sonoff Pow (±1%), é possível perceber que nos dois experimentos os servidores consumiram a mesma quantidade de energia. Pode-se considerar que isso é um indicativo de que os dois wattímetros monitoram o consumo energético corretamente, o que muda é a exatidão. Sendo que o Sonoff Pow é mais exato e, dessa forma, ele foi escolhido para monitorar o consumo energético dos servidores, no restante dos experimentos. É oportuno ressaltar que o único servidor que ficou fora da margem de erro, por uma ligeira diferença, foi o servidor computação1. Os dados também são expostos na Tabela 8.

Tabela 8 – Consumo Enerético por Servidor e Wattímetro durante 16 horas

Experimento	**Consumo Energético (kWh)**				
	controlador	**computação1**	**computação2**	**computação3**	**Total**
Sonoff Pow	1,272	1,37	1,304	1,292	5,238
IPMI	1,200	1,200	1,200	1,200	4,800

Figura 22 – Consumo Energético por Servidor e Wattímetro



6.1.2 Consumo Energético de Servidores Ociosos, Suspensos e Desligados

Observou-se que no experimento em que os servidores ficaram ociosos, o consumo energético foi considerável comparado com o experimento no qual os servidores ficaram sobrecarregados. O consumo total da nuvem com os servidores sobrecarregados foi de 6,974 kWh. Ao passo que no experimento em que os servidores ficaram ociosos, o consumo foi de 4,612 kWh. Isso representa um consumo energético de aproximadamente 66,131% do máximo consumido. Dessa forma, do ponto de vista do consumo energético, realmente não compensa manter servidores ociosos. Esse resultado também mostra que o consumo energético de servidores ociosos pode estar diminuindo, pois conforme (CIMA et al., 2015) e (BARROSO; CLIDARAS; HöLZLE, 2013) servidores ociosos consumiriam cerca de 80% de energia comparado com servidores sobrecarregados.

Além disso, servidores no modo suspenso também apresentaram elevado consumo energético, comparados com servidores sobrecarregados. No experimento em que os servidores foram colocados no modo suspenso eles consumiram um total de 4,301 kWh. Isso representa um consumo energético de aproximadamente 61,672% do máximo consumido. Este elevado consumo energético dos servidores no modo suspenso, provavelmente se deve ao fato que servidores não são projetados para ficarem neste modo. Eles são projetados para ficarem ligados o tempo todo. Segundo (BARROSO; CLIDARAS; HöLZLE, 2013), os fabricantes vêm buscando

melhorar a eficiência energética destes equipamentos. Porém, essa otimização é complexa, e requer melhorias em *hardware* e sistemas operacionais, por exemplo.

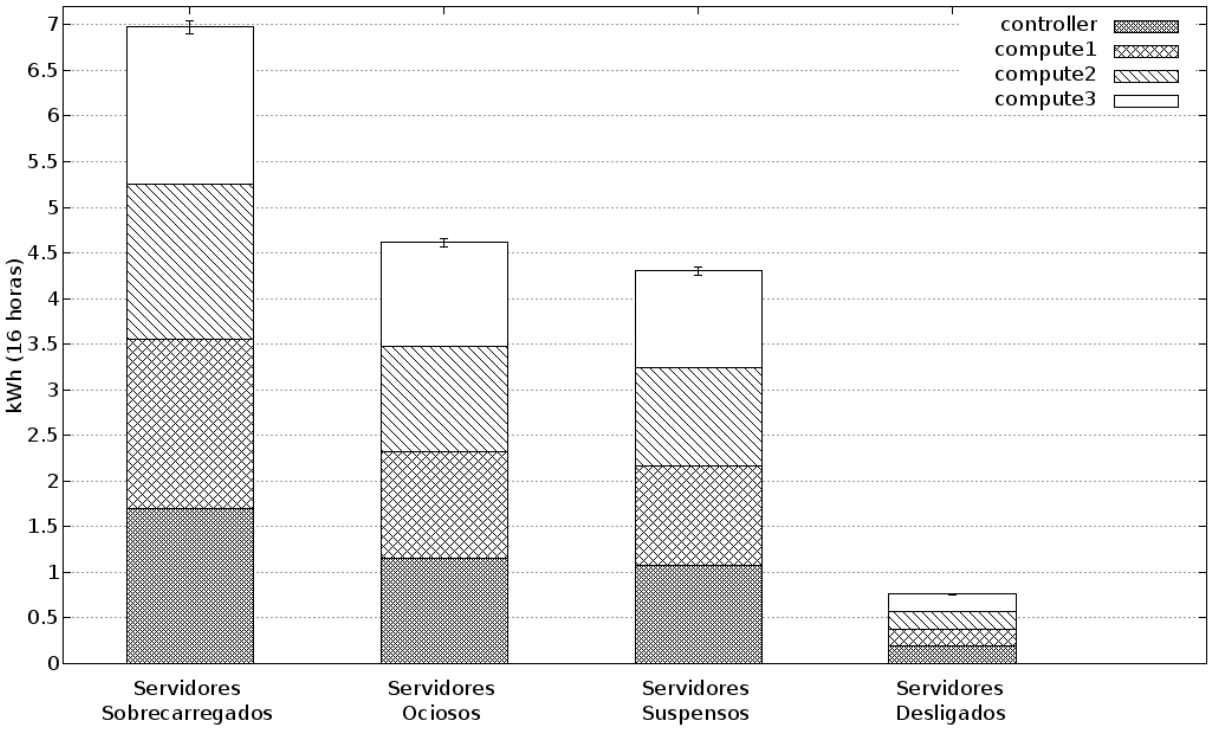
Além do elevado consumo energético de servidores ociosos e suspensos, pode-se observar que mesmo desligados, os servidores da nuvem apresentaram um consumo energético que não pode ser desprezado. O consumo, durante as 16 horas do experimento em que os servidores ficaram desligados, foi de 0,760 kWh. Isso representa 10,898% do consumo quando estão sobrecarregados. Isso ocorre pelo fato que a iDrac6 (IPMI) fica ativa, mesmo com o sistema operacional desligado. Esse resultado diverge do valor apresentado por (ORGERIE; LEFEVRE; GELAS, 2008), em que os autores apontam que o consumo energético de servidores desligados é de aproximadamente 5%. Além disso, é importante salientar que o consumo energético apresentado nesses experimentos é de servidores comerciais. Assim, podem existir servidores com *hardware* específico que podem consumir menos energia.

Dessa forma, para conseguir economizar uma quantidade maior de energia, optou-se por desligar os servidores nos experimentos de validação do arcabouço. Porém, é importante ressaltar que isso pode impactar de forma negativa no desempenho da nuvem, tendo em vista que servidores podem demorar um tempo considerável para serem ligados. Assim, a resposta a um aumento brusco de carga de trabalho, pode demorar mais que o desejável. Dessa forma, é importante buscar entender melhor a relação entre consumo energético, tempo de ligamento de servidores e desempenho da nuvem computacional. A Tabela 9 apresenta o consumo por servidor e por experimento. Para facilitar a visualização dos dados, é apresentada a Figura 23.

Tabela 9 – Consumo Energético de Servidores Ociosos, Suspensos e Desligados

Experimento	**Consumo Energético (kWh)**				
	controlador	**computação1**	**computação2**	**computação3**	**Total**
Servidores Sobrecarregados	1,695	1,863	1,700	1,716	6,974
Servidores Ociosos	1,153	1,175	1,151	1,133	4,612
Servidores Suspensos	1,077	1,092	1,076	1,056	4,301
Servidores Desligados	0,189	0,188	0,192	0,191	0,760

Figura 23 – Consumo Energético de Servidores Sobrecarregados, Ociosos, Suspensos e Desligados



6.2 Experimentos de Validação do Arcabouço

Esta subseção apresenta os resultados dos experimentos que foram utilizados para validação da eficiência do arcabouço, além do impacto causado por ele na nuvem computacional. Realizou-se o primeiro experimento sem migrações de VMs, que é o modo normal de operação do OpenStack. Esse experimento serviu como controle. Do segundo ao décimo sexto experimento, o robô Wall-E estava em operação. Assim, foi possível ocorrer migrações de VMs, ligamentos e desligamentos de servidores. Os resultados dos experimentos são apresentados nas próximas subseções e é possível observar o consumo energético de cada um dos servidores, latência das requisições HTTP, quantidade de requisições HTTP atendidas e não atendidas, quantidade de migrações e média de utilização do *hardware*.

6.2.1 Consumo Energético

A Tabela 10 apresenta o consumo energético durante os dezesseis experimentos. Além do consumo energético de cada servidor e consumo total da nuvem, também pode-se observar os limites aplicados. Durante as dezesseis horas do primeiro experimento, os quatro servidores consumiram um total de 5,779 kWh. Em comparação com esse experimento de controle, os experimentos que mais economizaram energia foram: ***(i)*** décimo quinto com um consumo de

3,524 kWh, **(*ii*)** décimo sexto experimento com um consumo de 3,541 kWh e o **(*iii*)** décimo quarto com um consumo de 3,566 kWh. Respeitando a margem de erro do wattímetro Sonoff Pow (±1%), pode-se afirmar que esses três experimentos consumiram a mesma quantidade de energia. Isso representa uma economia energética de aproximadamente 38%.

Em *data centers* de larga escala, a economia de energia pode ultrapassar os 38%. Para validar está possibilidade, há necessidade de realizar novos experimentos em um ambiente com maior número de servidores. Nestes três experimentos, em que houve uma maior economia de energia, o Limite Superior foi de 90%, o Limite de Segurança foi de 80%, e o Limite Inferior teve variação de 50, 60 e 70%. Esses foram os maiores limites aplicados durante os experimentos. Dessa forma, percebe-se que quanto maior forem os limites, maior é a economia de energia. Apesar do Limite Inferior ser diferente, nestes experimentos, este limite não influenciou no consumo energético. Isso se deve às intensidades de cargas aplicadas na nuvem computacional. Se as intensidades de cargas fossem maiores, muito provavelmente o consumo energético destes três experimentos não seria o mesmo. O experimento em que o Limite Inferior foi configurado como 70% provavelmente consumiria menos energia comparado com os experimentos em que o Limite Inferior foi configurado como 50 e 60%. E o experimento em que o Limite Inferior foi configurado como 50% consumiria mais energia comparado com os experimentos em que o Limite Inferior foi configurado como como 60 e 70%.

Os experimentos que não apresentaram economia de energia em comparação com o experimento de controle, foram: **(*i*)** segundo experimento com um consumo de 5,81 kWh, **(*ii*)** décimo experimento com um consumo de 5,812 kWh e **(*iii*)** quinto experimento com um consumo de 5,819 kWh. Percebe-se que nestes experimentos o Limite Inferior foi configurado como 10%. Isso impossibilitou ao robô Wall-E classificar os servidores como ociosos ou quase ociosos e consequentemente também impossibilitou a realização de migrações de VMs e desligamento de servidores. Isso ocorreu, pois, os servidores nunca tinham uma utilização de *hardware* menor que 10%, mesmo durante a carga da noite. Apesar desses três experimentos apresentarem um consumo maior de energia em comparação com o experimento de controle, se respeitada a margem de erro, pode-se dizer que todos os experimentos consumiram a mesma quantidade de energia.

O restante dos experimentos apresentaram economia de energia, em comparação ao experimento de controle. A economia variou entre 5,98% (terceiro experimento) e 33,25% (décimo terceiro experimento). Percebe-se que a medida em que os limites vão aumentando, o consumo energético total da nuvem diminuiu. Além disso, é importante ressaltar que em *data*

centers onde os servidores trabalhem com cargas intensivas durante todo tempo, o arcabouço muito provavelmente não conseguirá economizar energia. Pois os servidores estarão sobrecarregados a todo o momento.

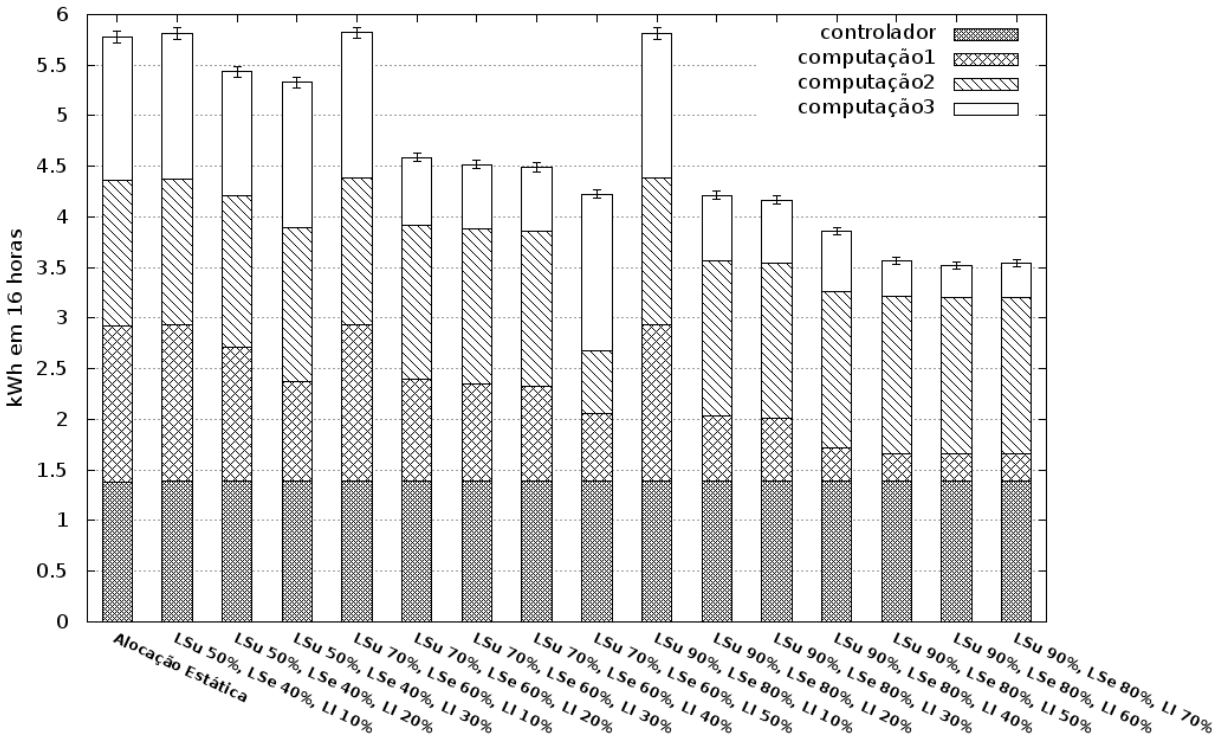
Tabela 10 – Limites de Utilização do *Hardware* e Consumo Energético dos Servidores por Experimento

Exp.	Limites(%)			Consumo Energético (kWh)				
	LSu	LSe	LI	controlador	computação1	computação2	computação3	Total
1 [a]	- [b]	- [c]	- [d]	1,385	1,538	1,434	1,422	5,779
2	50	40	10	1,393	1,546	1,440	1,431	5,810
3	50	40	20	1,396	1,320	1,489	1,228	5.433
4	50	40	30	1,397	0,977	1,521	1,434	5,329
5	70	60	10	1,392	1,545	1,445	1,437	5,819
6	70	60	20	1,395	1,001	1,526	0,667	4,589
7	70	60	30	1,392	0,959	1,532	0,637	4,521
8	70	60	40	1,397	0,931	1,536	0,625	4,489
9	70	60	50	1,397	0,667	0,619	1,542	4,225
10	90	80	10	1.392	1.543	1.446	1.431	5.812
11	90	80	20	1,393	0,642	1,531	0,648	4,214
12	90	80	30	1,395	0,620	1,527	0,623	4,165
13	90	80	40	1,395	0,323	1,542	0,598	3,858
14	90	80	50	1,393	0,273	1,548	0,352	3,566
15	90	80	60	1,396	0,270	1,539	0,319	3,524
16	90	80	70	1,395	0,271	1,538	0,337	3,541

[a] Alocação estática, com posicionamento inicial das VMs (subseção 5.2.2)
[b, c, d] Não se aplica

A Figura 24 apresenta os dados da Tabela 10 plotados. Nela, também é possível observar os limites utilizados, o consumo energético de cada um dos servidores e o consumo total da nuvem durante os dezesseis experimentos.

Figura 24 – Consumo Energético dos Servidores por Experimento



6.2.2 Desempenho da Nuvem Computacional

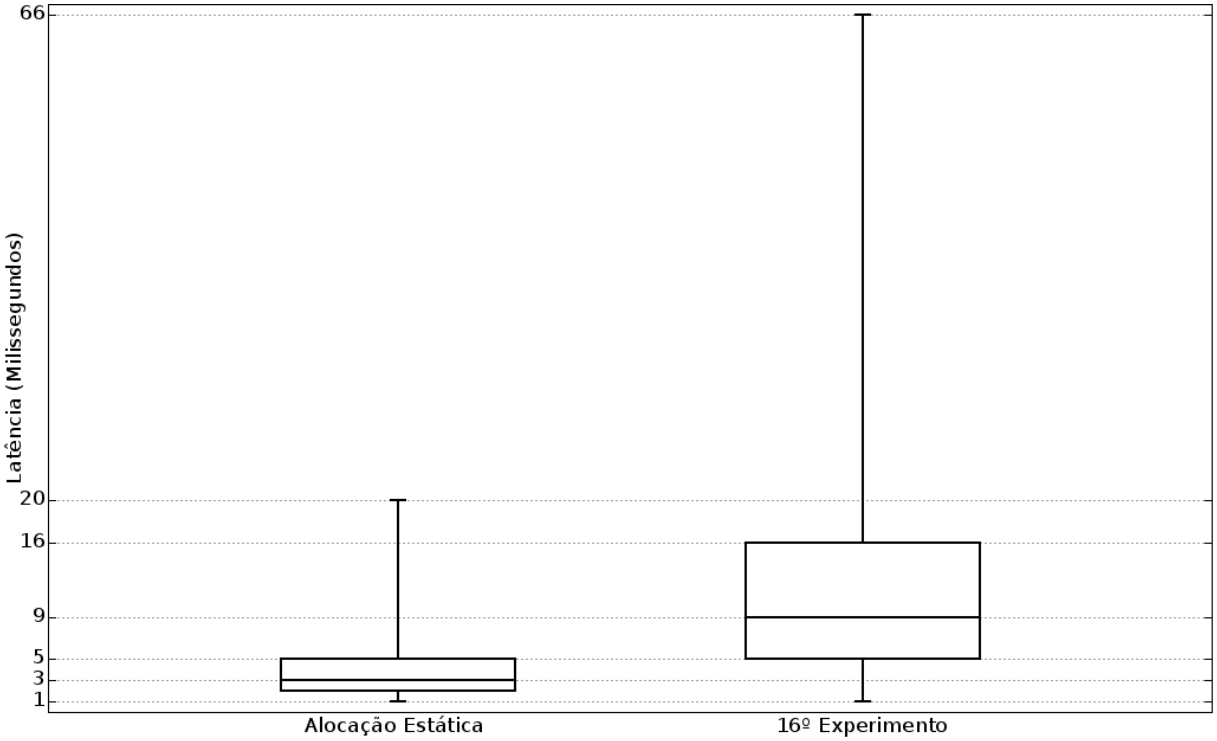
Para avaliar o desempenho da nuvem computacional, utilizou-se as seguintes métricas: **(*i*)** latência média das requisições HTTP, **(*ii*)** quantidade de requisições HTTP não atendidas e **(*iii*)** quantidade de requisições HTTP atendidas. Todas essas métricas são avaliadas, separadamente durante as cargas do dia e da noite. A Tabela 11 apresenta a latência média das requisições, requisições atendidas e não atendidas durante as cargas do dia. É oportuno ressaltar, que cada experimento teve dois períodos com cada intensidade de carga. Assim, cada intensidade teve duração total de oito horas.

Latência Média das Requisições HTTP Durante as Cargas do Dia

Durante o primeiro experimento, que serviu de controle e que não ocorreram migrações, a latência média das requisições HTTP, durante a carga do dia, foi de 5,095 milissegundos. Os três experimentos que mais economizaram energia também foram os experimentos que tiveram o maior aumento na latência média. No décimo sexto experimento, por exemplo, o aumento na latência foi de aproximadamente 8 milissegundos. Tendo uma latência média de 13,030 milissegundos. Apesar da latência média ser uma métrica de análise importante, ela pode esconder comportamentos importantes. Assim, calculou-se a dispersão dos dados do experimento com alocação estática de VMs e do décimo sexto experimento, que foi o experimento que apresentou a maior economia de energia e o maior aumento da latência média.

No experimento com alocação estática, durante as oito horas com carga do dia, 50% das requisições HTTP apresentaram latência entre 2 e 5 milissegundos. Ampliando essa porcentagem de requisições para 99% a latência variou entre 1 e 20 milissegundos e 1% das requisições apresentaram latência entre 20,1 e 199 milissegundos. A latência mediana apresentada foi de 3 milissegundos. Já no décimo sexto experimento, durante as oito horas com carga do dia, 50% das requisições HTTP apresentaram latência entre 5 e 16 milissegundos. Ampliando essa porcentagem de requisições para 99% a latência variou entre 1 e 66 milissegundos e 1% das requisições apresentou latência entre 66,1 e 199 milissegundos. A latência mediana apresentada foi de 9 milissegundos. Dessa forma, o aumento na latência mediana do décimo sexto experimento comparado com o experimento de controle foi de apenas 6 milissegundos. A Figura 25 apresenta esses valores plotados, em que é possível visualizar os quartis, que delimitam 50% das mensagens e as hastes que delimitam 99% das mensagens.

Figura 25 – Dispersão da Latência das Requisições HTTP



Esse aumento na latência durante o experimento décimo sexto ocorreu devido aos seguintes fatores: os limites eram maiores comparados aos limites dos outros experimentos, fazendo com que mesmo durante a primeira carga do dia, o robô Wall-E classificasse os servidores como quase ociosos. Assim, as migrações começaram já no início do experimento. Dessa forma, as VMs ficaram consolidadas em apenas um servidor de computação a maior parte do experimento, ao contrário do experimento de controle, em que as VMs ficaram distribuídas nos três servidores de computação durante todo tempo. Além disso, as migrações ocorreram durante a carga mais intensa. Migrações que ocorrem durante cargas com certa intensidade, podem degradar consideravelmente o desempenho da VM. Isso ocorre porque a VM pode gravar em páginas de memória mais rapidamente do que elas podem ser copiadas entre os servidores de computação. Quando isso ocorre, o Nova determina que a migração é impossível de acontecer e para que a migração seja viável, desacelera a CPU da VM até que o processo de cópia de memória seja mais rápido do que a gravação de memória (OPENSTACK, 2018a).

Além disso, perto do final da cópia da memória, a VM é pausada por um curto período de tempo para que as poucas páginas restantes possam ser copiadas sem a interferência das

gravações de memória. Esse tempo geralmente é em torno de 50 milissegundos, dependendo da capacidade de cada VM. Quando o Nova ainda percebe que a cópia da memória está sendo mais lenta que a gravação na memória, ele aumenta este tempo gradualmente até que a migração ocorra com sucesso (OPENSTACK, 2019).

Apesar do arcabouço ter aumentado a latência média das requisições, o maior aumento durante os experimentos ($\approx$ 8 milissegundos), está longe de ser perceptível para os humanos. Segundo Nielsen (1993) usuários não conseguem perceber latência menor ou igual 100 milissegundos. Quando a latência é menor ou igual a este limiar, usuários têm a percepção de que o retorno da aplicação é instantâneo. Porém, é importante salientar que o usuário pode ser outra aplicação e não um humano. Assim, a latência, mesmo que imperceptível para humanos, pode ser indesejável para outras aplicações.

Além disso, em uma nuvem computacional em operação, essa latência causada pelo arcabouço pode ser somada a outras latências externas ao ambiente e, dessa forma, o arcabouço pode contribuir para a violação do SLA da nuvem. Pode-se citar como exemplos de latências externas ao *data center* a latência causada por rotas de menor qualidade e a latência devido à distância entre o *data center* e o usuário que acessa a aplicação.

Tabela 11 – Latência Média, Requisições Atendidas e Não atendidas (HTTP) Durante as Cargas do Dia

Experimento	**Limites (%)**			**Latência**	**Requisições**	
	LSu	**LSe**	**LI**	**Média (ms)**	**Atendidas**	**Não Atendidas**
1 [a]	- [b]	-[c]	-[d]	5,095	7.603.183	0
2	50	40	10	5,078	7.603.088	0
3	50	40	20	5,515	7.560.010	6
4	50	40	30	6,982	7.451.294	4
5	70	60	10	5,104	7.601.351	0
6	70	60	20	11,245	7.234.725	2
7	70	60	30	11,381	7.221.249	4
8	70	60	40	12,317	7.182.807	3
9	70	60	50	10,849	7.237.960	8
10	90	80	10	5,083	7.606.198	0
11	90	80	20	9,218	7.432.585	0
12	90	80	30	9,047	7.445.208	17
13	90	80	40	9,925	7.397.279	15
14	90	80	50	12,099	7.339.118	8
15	90	80	60	12,516	7.287.178	9
16	90	80	70	13,030	7.243.115	7

[a] Alocação estática, com posicionamento inicial das VMs (subseção 5.2.2)
[b, c, d] Não se aplica

Requisições HTTP não Atendidas durante as Cargas do Dia

O experimento de controle não apresentou perda de requisições durante as cargas do dia. Devido a pausas nas VMs, para finalização das migrações, quase todos os outros experimentos apresentaram perdas de requisições HTTP. Essas perdas muito provavelmente ocorreram devido ao TTL (*Time to Live*) ter expirado. O TTL das requisições HTTP no JMter é de 200 milissegundos, conforme documentação oficial (APACHE, 2019a). Os dois experimentos que apresentaram maiores perdas foram os experimentos décimo segundo e décimo terceiro. O décimo segundo experimento, por exemplo, apresentou uma perda de 17 mensagens. Isso corresponde a aproximadamente 0,00023% do total de requisições atendidas. Apesar de ser um percentual de pequena escala, isso pode causar um impacto negativo na percepção dos usuários mesmo que seja por breves períodos de tempo.

Requisições HTTP Atendidas Durante as Cargas do Dia

No experimento que serviu de controle, a nuvem computacional atendeu, durante as cargas do dia, um total de 7.603.183 requisições HTTP. O experimento que apresentou o menor número de requisições atendidas, foi o oitavo experimento. Esse experimento também foi um dos experimentos em que a latência foi maior. Analisando os dados da Tabela 11 percebe-se que há uma relação direta entre requisições atendidas e latência média. A tendência é que quanto maior a latência, menor é a quantidade de requisições HTTP atendidas. Isso se deve ao fato que o JMeter foi configurado para esperar a resposta da requisição HTTP ou expiração do TLL antes de fazer outra requisição para o mesmo arquivo.

Análise das Métricas Durante as Cargas da Noite

A Tabela 12 apresenta a latência média das requisições, requisições atendidas e não atendidas durante as cargas da noite. O experimento que serviu de controle apresentou latência média de 2,656 milissegundos. Da mesma forma que a consolidação de VMs impactou na latência das requisições HTTP durante as cargas do dia, ela também impactou durante as cargas da noite. Porém, como as cargas da noite eram de menor intensidade, este impacto foi menor

comparado com o impacto ocorrido nas cargas do dia. O experimento que apresentou maior aumento da latência média das requisições HTTP, comparado com o experimento de controle, foi o nono experimento. Nesse experimento, o aumento foi de 1,162 milissegundo. Apesar de que qualquer aumento na latência não seja desejável, pode-se considerar como um aumento mínimo e que dificilmente trará grandes impactos.

Além disso, apenas o experimento de controle apresentou requisição não atendida. Nele, uma requisição não foi atendida. Não se conseguiu fazer relação dessa perda com outro fator. Já as requisições atendidas, durante as cargas da noite, apresentaram pouca variação. Sendo que no experimento de controle, a nuvem computacional atendeu um total de 86.333 mensagens. E o experimento que apresentou menos requisições atendidas foi o décimo primeiro, com um total de 86.307 mensagens. A Figura 26 apresenta a latência média das requisições HTTP por experimento durante as cargas do dia e da noite. A Figura 27 apresenta a quantidade de requisições HTTP atendidas por experimento durante as cargas do dia e da noite. Já a Figura 28 apresenta a soma das requisições HTTP não atendidas por experimento durante as cargas do dia e da noite.

Tabela 12 – Latência Média, Requisições Atendidas e Não atendidas (HTTP) Durante Carga da Noite

Experimento	**Limites (%)**			**Latência**	**Requisições**	
	LSu	**LSe**	**LI**	**Média (ms)**	**Atendidas**	**Não Atendidas**
1 [a]	- [b]	- [c]	- [d]	2,656	86.333	1
2	50	40	10	2,624	86.328	0
3	50	40	20	3,000	86.322	0
4	50	40	30	3,354	86.282	0
5	70	60	10	2,646	86.329	0
6	70	60	20	3,738	86.307	0
7	70	60	30	3,507	86.312	0
8	70	60	40	3,666	86.310	0
9	70	60	50	3,818	86.307	0
10	90	80	10	2,617	86.326	0
11	90	80	20	3,488	86.307	0
12	90	80	30	3,735	86.315	0
13	90	80	40	3,406	86.314	0
14	90	80	50	3,191	86.319	0
15	90	80	60	3,085	86.317	0
16	90	80	70	3,130	86.314	0

[a] Alocação estática, com posicionamento inicial das VMs (subseção 5.2.2)
[b, c, d] Não se aplica

Figura 26 – Latência Média das Requisições HTTP por Experimento

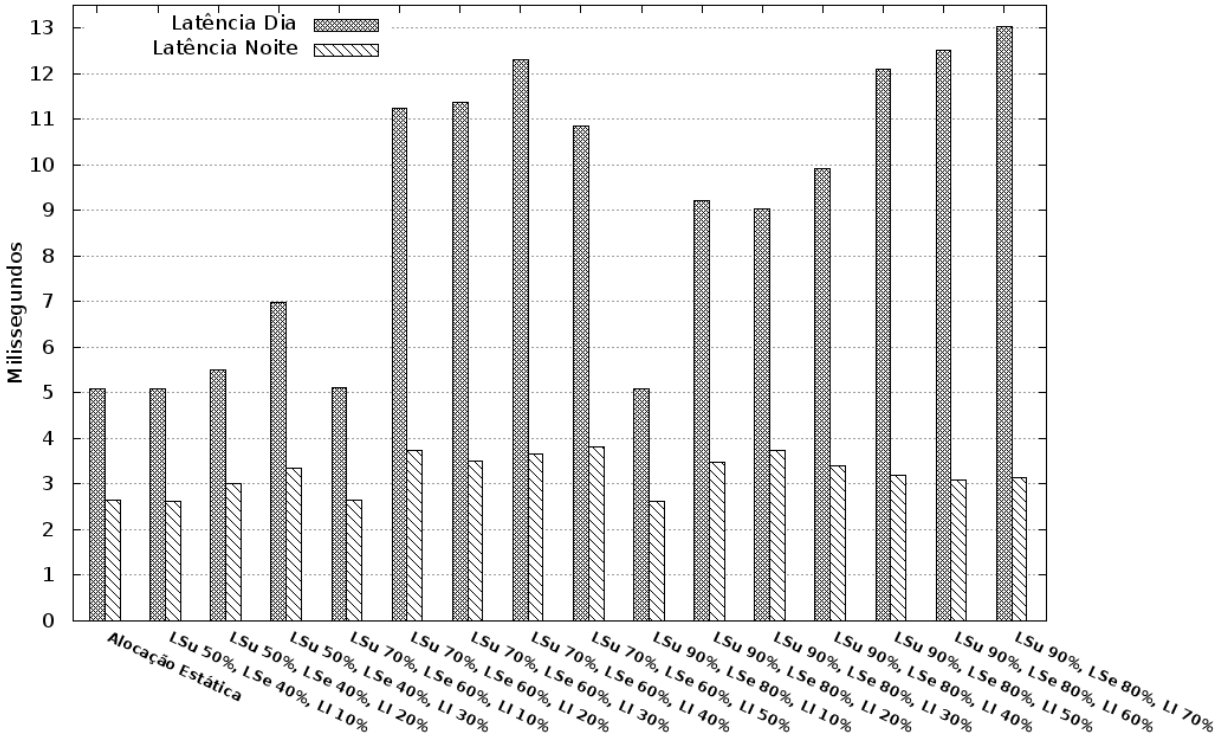


Figura 27 – Requisições HTTP Atendidas por Experimento

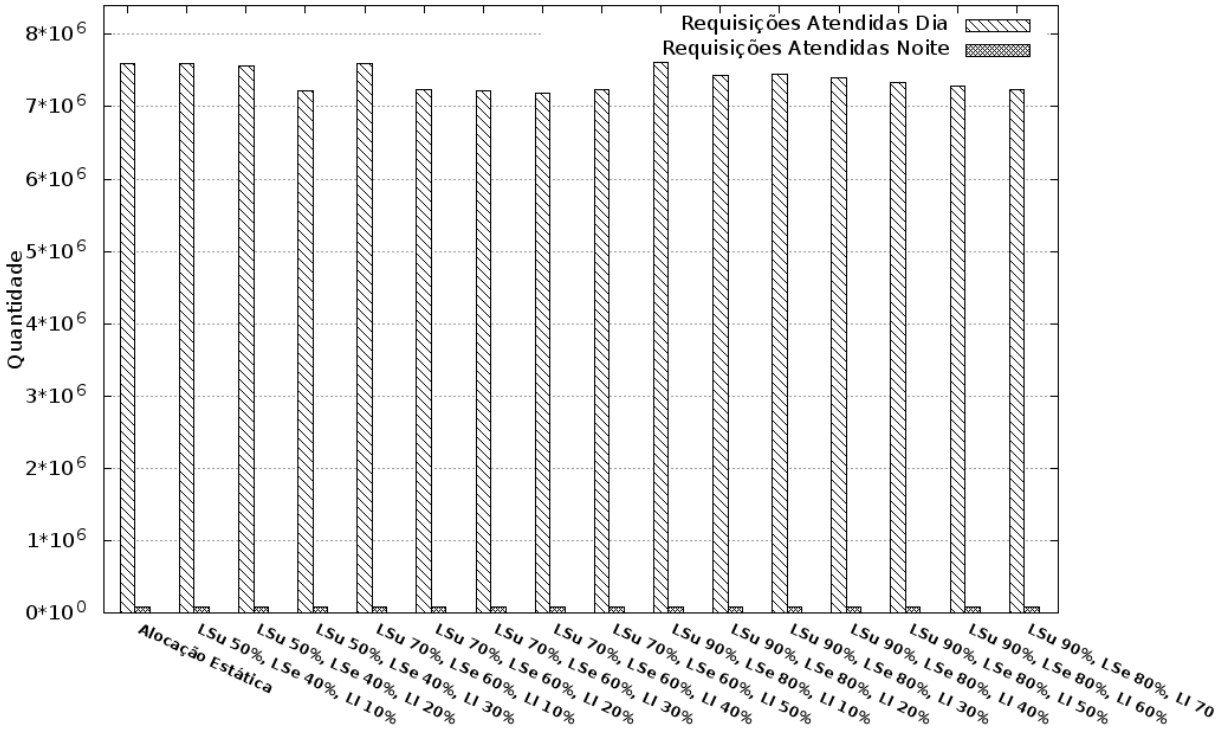
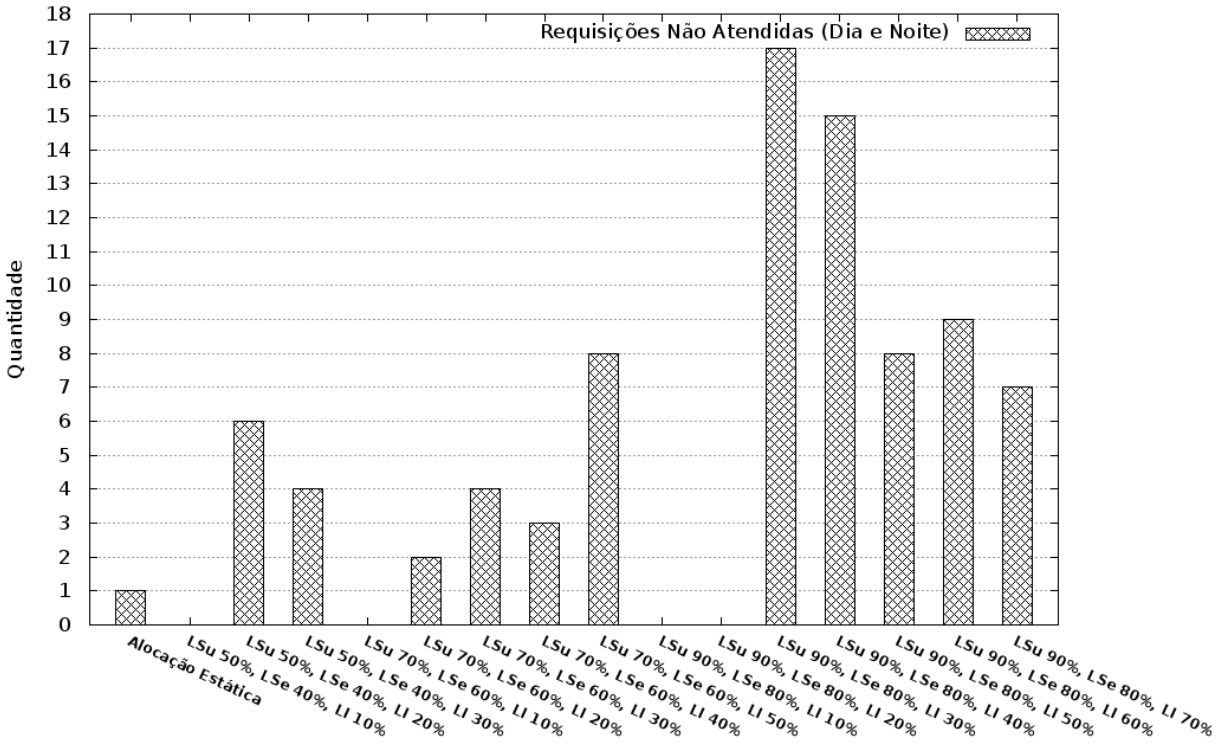


Figura 28 – Requisições Não Atendidas por Experimento



6.2.3 Migrações

A quantidade de migrações está diretamente relacionada aos limites aplicados, conforme é possível observar na Tabela 13. Nos experimentos em que o limite inferior foi configurado como 10% (segundo, quinto e décimo experimentos), não ocorrerem migrações. Isso aconteceu, pois mesmo durante as cargas da noite, a utilização do *hardware* dos servidores nunca eram menores que este percentual. Assim, o robô Wall-E não classificava os servidores como quase ociosos e assim não ocorriam migrações. Além disso, durante as cargas do dia, todos os servidores ficavam classificados como normais ou quase sobrecarregados, e da mesma forma, as migrações não ocorriam.

Os experimentos que apresentaram as maiores quantidades de migrações foram os experimentos em que o limite superior foi configurado como 70% e o limite de segurança foi configurado como 60%. Isso ocorreu pois, durante as cargas do dia, as VMs ficavam em dois ou três servidores e a noite elas ficavam hospedadas em apenas um servidor. Isso fez com que ocorressem migrações todas as vezes que a intensidade da carga era alterada. Já os experimentos em que o limite superior foi configurado como 90% e o limite de segurança foi configurado como 80%, na maioria dos experimentos ocorreram 20 migrações. Isso aconteceu, pois já de início

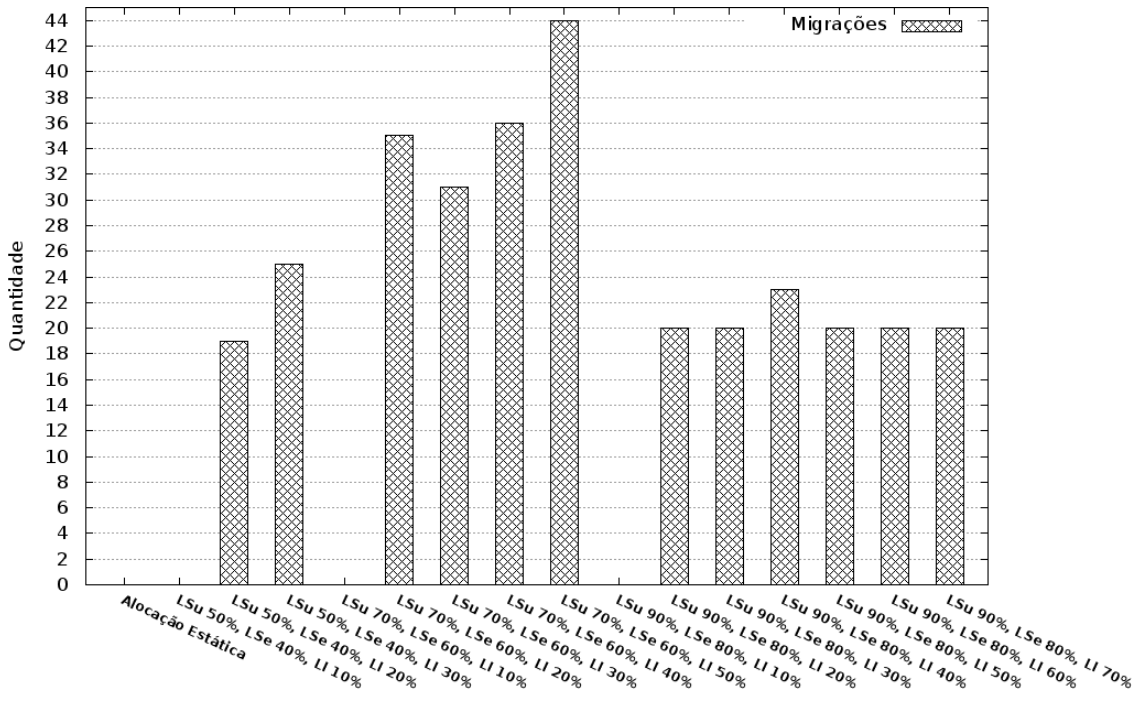
as migrações já começaram a ser realizadas, e as VMs ficaram a maior parte dos experimentos hospedadas em apenas um servidor. Pois, mesmo durante as cargas do dia, os servidores eram classificados como quase ociosos ou normais. A Tabela 13 apresenta a quantidade de migrações por experimento. Já a Figura 29 apresenta os dados plotados.

Tabela 13 – Quantidade de Migrações por Experimento

Experimento	**Limites (%)**			**Quantidade de Migrações**
	LSu	LSe	LI	
1 [a]	- [b]	- [c]	- [d]	0
2	50	40	10	0
3	50	40	20	19
4	50	40	30	25
5	70	60	10	0
6	70	60	20	35
7	70	60	30	31
8	70	60	40	36
9	70	60	50	44
10	90	80	10	0
11	90	80	20	20
12	90	80	30	20
13	90	80	40	23
14	90	80	50	20
15	90	80	60	20
16	90	80	70	20

[a] Alocação estática
[b, c, d] Não se aplica

Figura 29 – Migrações por Experimento



6.2.4 Utilização do Hardware

Assim como a economia de energia, desempenho da nuvem computacional e quantidade de migrações, a utilização do *hardware* também está diretamente relacionada com os limites aplicados nos experimentos. A tendência é que, quanto maior forem os limites, maior é a utilização do *hardware*. Isso pode ser observado na Tabela 14, em que são apresentados os limites aplicados em cada experimento, a utilização média da CPU e RAM por servidor, além da média de utilização da CPU e RAM da nuvem, que é a média de utilização dos 4 servidores.

No primeiro experimento, que serviu de controle, a nuvem apresentou uma média de utilização de CPU de 43,186% e de 26,208% de RAM. O segundo, o quinto e o décimo experimentos apresentaram praticamente a mesma utilização de *hardware* do primeiro experimento. Nesses experimentos, o limite inferior foi configurado como 10%. Isso ocorreu pelo fato de que durante as cargas do dia, os servidores ficavam classificados como normais ou quase sobrecarregados e durante as cargas da noite, a utilização do *hardware* nunca era menor que o limite inferior. Impossibilitando ao robô Wall-E de realizar migrações de VMs. Nos demais experimentos, de forma geral, percebe-se que quanto maiores são os limites, maior é a utilização do *hardware*. O maior aumento de utilização da CPU da nuvem foi no experimento décimo segundo que teve uma utilização média de 56,537%. E o maior aumento de utilização da RAM da nuvem foi no experimento décimo terceiro, que apresentou uma utilização média de 32,654%.

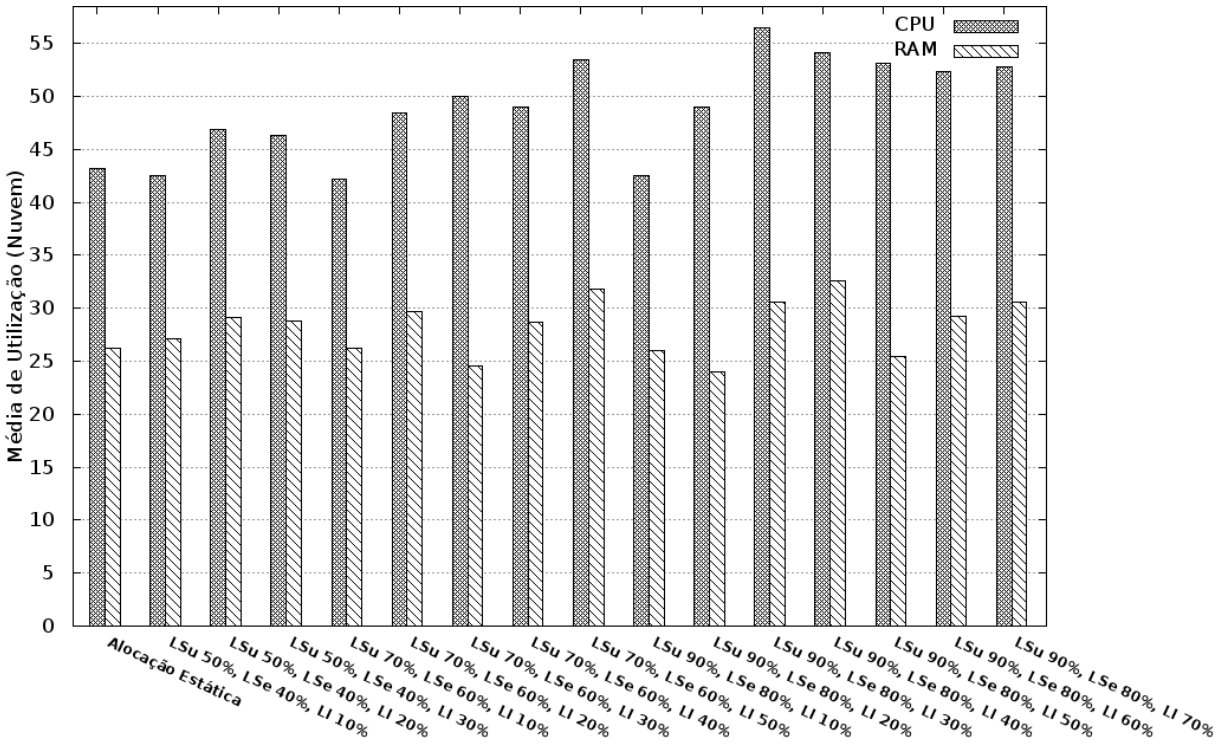
A nuvem realizou o mesmo trabalho durante todos os experimentos. Dessa forma, esperava-se que a média de utilização de *hardware* da nuvem durante os experimentos sofresse pouca ou nenhuma variação. As hipóteses mais prováveis para justificar este aumento na utilização do *hardware* da nuvem, são as migrações e o *overhead* da virtualização. As migrações adicionam carga extra nos servidores, pois há cópia de informações entre os servidores de computação e o servidor controlador precisa supervisionar a tarefa. Já o *overhead* da virtualização, pode ter um comportamento exponencial, que cresce de acordo com a quantidade de VMs hospedadas em um mesmo servidor, como sugerem os pesquisadores (TONG et al., 2011). Essas hipóteses precisam ser comprovadas, pois podem ajudar na tomada de decisões sobre as migrações. A Figura 30 apresenta a utilização média de CPU e RAM da nuvem computacional por experimento.

Tabela 14 – Média de Utilização do Hardware por Experimento

Exp.	Limites (%)			Média de Utilização do Hardware (%)									
				controlador		computação1		computação2		computação3		Nuvem	
	LSu	LSe	LI	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM	CPU	RAM
1 [a]	- [b]	- [c]	- [d]	9,237	32,993	53,934	25,460	54,758	22,690	54,816	23,690	43,186	26,208
2	50	40	10	9,784	37,037	54,202	23,646	52,873	24,237	53,314	23,465	42,543	27,096
3	50	40	20	10,019	37,981	62,027	26,788	59,619	29,987	56,071	21,822	46,934	29,144
4	50	40	30	11,598	39,051	57,701	17,205	64,877	37,963	51,234	21,113	46,352	28,833
5	70	60	10	9,659	34,370	52,902	23,382	53,085	23,947	53,260	23,183	42,226	26,220
6	70	60	20	9,733	34,110	52,377	16,649	69,215	45,867	62,646	22,370	48,493	29,749
7	70	60	30	9,744	24,370	53,846	14,288	69,741	41,069	66,707	18,383	50,009	24,527
8	70	60	40	9,875	30,452	46,753	13,627	72,238	49,772	66,957	20,721	48,956	28,643
9	70	60	50	9,939	33,050	49,395	14,967	81,345	31,105	73,362	48,087	53,510	31,802
10	90	80	10	9,616	35,070	53,586	22,583	54,371	23,718	52,655	22,485	42,557	25,964
11	90	80	20	9,744	24,970	52,546	13,380	67,941	40,085	65,707	17,452	48,984	23,972
12	90	80	30	9,431	28,432	77,166	20,800	70,655	47,697	68,898	25,319	56,537	30,562
13	90	80	40	9,376	35,426	63,415	17,429	74,063	56,323	69,804	21,437	54,164	32,654
14	90	80	50	9,24	25,565	58,727	12,487	74,688	48,909	70,078	14,914	53,183	25,469
15	90	80	60	9,217	27,751	56,795	14,838	74,765	57,048	68,494	17,547	52,318	29,296
16	90	80	70	9,284	29,013	57,618	15,672	74,740	58,093	69,737	19,417	52,845	30,549

[a] Alocação estática, com posicionamento inicial das VMs (subseção 5.2.2)
[b, c, d] Não se aplica

Figura 30 – Média de Utilização do *Hardware* por Experimento



7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, apresentou-se um arcabouço que teve como meta principal a economia de energia em *data centers* com VMs e que a economia interferisse minimamente no desempenho destes centros. Para conseguir atingir a meta, o arcabouço foi capaz de propiciar o monitoramento centralizado do consumo energético e carga de trabalho de servidores, classificação de servidores com base nos dados do monitoramento, alocação dinâmica de VMs com migração a quente e desligamento de servidores ociosos.

O arcabouço foi validado em ambiente real, utilizando uma nuvem computacional implementada com o OpenStack. Foram executados vários experimentos, nos quais foi possível observar uma economia de energia de até 38%, com degradação do desempenho aparentemente imperceptível à humanos. Apesar do arcabouço ter propiciado considerável economia de energia, a economia está diretamente associada à carga de trabalho e quantidade de servidores. Em *data centers* onde as cargas são intensas durante todo tempo, por exemplo, o arcabouço poderá economizar pouca ou até mesmo nenhuma energia e, em *data centers* de larga escala, onde as cargas são sazonais, a economia pode ultrapassar os 38%.

Além da considerável economia de energia, o trabalho apresenta as seguintes contribuições: ***(i)*** desenvolvimento dos *middlewares* kWh-MONITOR, HW-SNMP, MODELOS-SYNC, MAPPER-VMS, ***(ii)*** desenvolvimento de metodologia para quantificação da utilização do *hardware* de servidores, ***(iii)*** desenvolvimento de metodologia de classificação de servidores com base na utilização do *hardware* e ***(iv)*** desenvolvimento de metodologia de decisão para migração de VMs, ligamento e desligamento de servidores.

Outrossim, conforme experimentos realizados, ficou comprovado que o modo de economia sugerido por BELOGLAZOV; BUYYA mostrou-se pouco eficiente. Servidores no modo suspenso apresentam consumo energético elevado, quando comprado com servidores ligados. Além disso, a política sugerida por BELOGLAZOV; BUYYA e MELHEM et al., possui notável problema. O problema são as migrações desnecessárias causadas pela metodologia de classificação dos servidores. Como exemplo, pode-se citar a seguinte situação: um servidor com utilização de CPU ligeiramente menor que o limite superior será classificado como normal. Assim, ficará passível de receber novas VMs de outros servidores classificados como ociosos. Dessa forma, se esse servidor receber novas VMs, provavelmente passará a ser classificado como sobrecarregado. Fazendo com que seja necessário migrar algumas VM que acabou de receber.

A metodologia de classificação proposta neste trabalho resolve o problema citado, pois servidores classificados como quase sobrecarregados não podem receber VMs de outros servidores. Tendo em vista o elevado consumo de energia elétrica pelos *data centers*, essas são importantes contribuições. Porém, como esta é uma área de pesquisa com grandes desafios, pesquisadores precisam estar em constante inquietação, buscando sempre o avanço da ciência para otimização dos recursos. Como trabalhos futuros sugere-se:

- Durante os experimentos, os limites superior, de segurança e inferior foram escolhidos intuitivamente. Provavelmente, esse não é o melhor método de escolha. Implementar um sistema computacional inteligente para escolher dinamicamente os limites, pode equacionar melhor o problema de economia de energia versus degradação do desempenho. RNA e Lógica Fuzzy são possibilidades a serem levadas em consideração neste caso;
- Em caso de necessidade de migração de VMs, os servidores de origem e destino são escolhidos com base apenas na ordem de classificação. Porém, não se leva em consideração qual é o melhor servidor em cada faixa de classificação. Realizar esse ranqueamento pode melhorar a performance do arcabouço. Além de ranquear os servidores, outra possibilidade é realizar o ranqueamento das VMs e não apenas utilizar o nome do modelo para decidir qual VM migrar.
- A quantificação da taxa de utilização do *hardware* é feita utilizando a média aritmética da utilização da CPU e RAM das oito últimas medições. Utilizar a média exponencial móvel pode melhorar o desempenho do arcabouço. Pois esse tipo de média é um importante indicador de tendência. Dessa forma, evita-se que picos bruscos de processamento interfira de forma negativa no desempenho do arcabouço e o arcabouço responderia mais rapidamente a necessidades de migrações.
- Descentralizar os mecanismos para alocação dinâmica de VMs pode ser importante, tendo em vista a escalabilidade e resiliência do arcabouço. Uma das possibilidades, são os próprios servidores de computação informarem, para o arcabouço, qual é a taxa de utilização do *hardware*, ao invés do arcabouço monitorar cada um dos servidores e realizar esse cálculo.
- Apesar do aumento da latência média, observada nos experimentos, ser de pequena escala, é importante otimizá-la. Uma possível otimização, seria: em momentos de baixa carga de trabalho, ao invés de desligar todos os servidores ociosos, desligar a maior parte dos servidores ociosos e manter poucos servidores ociosos suspensos. Assim, a medida que a carga for aumentando, desperta-se servidores suspensos para receber VMs e liga-se servidores desligados e coloca-os em modo suspenso. Isso pode ser vantajoso, tendo em vista que o tempo de despertamento de um servidor suspenso é muito mais rápido comparado com o tempo de

ligamento de um servidor. Assim, tem-se a grande economia de energia que o desligamento proporciona e a rapidez do despertamento de servidores suspensos para hospedagem de VMs. Isso pode ser viável em um ambiente de larga escala;

- O kWh-PRO propicia facilidade de realização de testes, pois não há necessidade de trabalhar com várias ferramentas. Dessa forma, a realização de experimentos com outras plataformas *Open Source* de computação em nuvem, pode ser importante. Como exemplo pode-se citar: CloudStack, Eucalyptus e OpenNebula. Além disso, sugere-se ampliar o número de servidores para verificação da escalabilidade e resiliência do arcabouço.
- A carga extra, gerada pelas migrações, não foi quantificada. Entender isso pode ajudar na tomada de decisões. Porém, antes de quantificar esta carga, é importante entender se o comportamento do *overhead* gerado pelas VMs é linear ou exponencial de acordo com a quantidade de VMs hospedadas em um único servidor;
- Contêineres vem ganhando destaque devido ao pequeno *overhead* comparado com o *overhead* das VMs. Porém, a migração a quente ainda não é totalmente estável. Dessa forma, pesquisas nessa área são desafiadoras e promissoras.

REFERÊNCIAS BIBLIOGRÁFICAS

AHMAD, N.; KANWAL, A.; SHIBLI, M. A. Survey on secure live virtual machine (VM) migration in cloud. In: **2013 2nd National Conference on Information Assurance (NCIA)**. IEEE, 2013. Disponível em: <https://doi.org/10.1109/ncia.2013.6725332>.

AIKEMA, D. et al. Green cloud VM migration: Power use analysis. In: **2012 International Green Computing Conference (IGCC)**. IEEE, 2012. Disponível em: <https://doi.org/10.1109/igcc.2012.6322249>.

ALI, Q. et al. Power aware NUMA scheduler in VMware ESXi hypervisor. In: **2015 IEEE International Symposium on Workload Characterization**. IEEE, 2015. Disponível em: <https://doi.org/10.1109%2Fiiswc.2015.30>.

AMS, A. M. S. **Bin Packing**. 2018. Disponível em: <http://www.ams.org/publicoutreach/feature-column/fcarc-bins3>.

APACHE. **Apache JMeter**. 2018. Disponível em: <https://jmeter.apache.org/>.

APACHE. **Introduction JMeter**. 2018. Disponível em: <http://jmeter.apache.org/usermanual/component_reference.html>.

APACHE. **What is the Apache HTTP Server Project?** 2018. Disponível em: <https://httpd.apache.org/ABOUT_APACHE.html>.

APACHE. **19.13 Apache HTTP Components HTTP Client Configuration**. 2019. Disponível em: <https://jmeter.apache.org/usermanual/properties_reference.html>.

APACHE, T. S. F. **ab - Apache HTTP server benchmarking tool**. 2019. Disponível em: <https://httpd.apache.org/docs/2.4/programs/ab.html>.

ARROBA, P. et al. DVFS-aware consolidation for energy-efficient clouds. In: **2015 International Conference on Parallel Architecture and Compilation (PACT)**. Institute of Electrical and Electronics Engineers (IEEE), 2015. Disponível em: <http://dx.doi.org/10.1109/PACT.2015.59>.

AVRAM, C.-A.; SALEM, K.; WONG, B. Latency amplification: Characterizing the impact of web page content on load times. In: **2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops**. IEEE, 2014. Disponível em: <https://doi.org/10.1109/srdsw.2014.16>.

AVRAMOV, J. R. L. **Request for Comments: 8238**. 2019. Disponível em: <https://tools.ietf.org/html/rfc8238>.

AWASTHI, A.; GUPTA, R. Multiple hypervisor based open stack cloud and VM migration. In: **2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)**. Institute of Electrical and Electronics Engineers (IEEE), 2016. Disponível em: <http://dx.doi.org/10.1109/CONFLUENCE.2016.7508101>.

BABU, A. et al. System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver. In: **2014 Fourth International Conference on Advances in Computing and Communications**. IEEE, 2014. Disponível em: <http://dx.doi.org/10.1109/ICACC.2014.66>.

BARROSO, L. A.; CLIDARAS, J.; HöLZLE, U. The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition. **Synthesis Lectures on Computer Architecture**, Morgan & Claypool Publishers LLC, v. 8, n. 3, p. 1–154, jul 2013. Disponível em: <https://doi.org/10.2200/s00516ed2v01y201306cac024>.

BELOGLAZOV, A.; BUYYA, R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. **Concurrency and Computation: Practice and Experience**, Wiley-Blackwell, v. 24, n. 13, p. 1397–1420, oct 2012. Disponível em: <https://doi.org/10.1002/cpe.1867>.

BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. **IEEE Cloud Computing**, Institute of Electrical and Electronics Engineers (IEEE), v. 1, n. 3, p. 81–84, set 2014. Disponível em: <http://dx.doi.org/10.1109/MCC.2014.51>.

CACCIATORE, K. et al. **Exploring Opportunities: Containers and OpenStack**. 2015. Disponível em: <https://www.openstack.org/assets/pdf-downloads/Containers-and-OpenStack.pdf>.

CANONICAL. **What is LXC?** 2017. Disponível em: <https://linuxcontainers.org/lxc/introduction/>.

CANONICAL. **Ubuntu Product Month**. 2018. Disponível em: <https://www.ubuntu.com/>.

CANONICAL, U. L. **Man Page - Stress-ng**. 2018. Disponível em: <http://manpages.ubuntu.com/manpages/bionic/man1/stress-ng.1.html>.

CANONICAL, U. L. **Setting Up NFS How To**. 2018. Disponível em: <https://help.ubuntu.com/community/SettingUpNFSHowTo>.

CASE, J. et al. **Request for Comments: 1157**. 1990. Disponível em: <https://www.ietf.org/rfc/rfc1157.txt?number=1157>.

CIMA, V. et al. Adding energy efficiency to openstack. In: **2015 Sustainable Internet and ICT for Sustainability (SustainIT)**. IEEE, 2015. Disponível em: <https://doi.org/10.1109/sustainit.2015.7101358>.

COLEMAN, M. **CONTAINERS AND VMS TOGETHER**. 2018. Disponível em: <https://blog.docker.com/2016/04/containers-and-vms-together/>.

CORRADI, A.; FANELLI, M.; FOSCHINI, L. VM consolidation: A real case based on OpenStack cloud. **Future Generation Computer Systems**, Elsevier BV, v. 32, p. 118–127, mar 2014. Disponível em: <https://doi.org/10.1016/j.future.2012.05.012>.

DAYARATHNA, M.; WEN, Y.; FAN, R. Data center energy consumption modeling: A survey. **IEEE Communications Surveys & Tutorials**, Institute of Electrical and Electronics Engineers (IEEE), v. 18, n. 1, p. 732–794, 2016. Disponível em: <http://dx.doi.org/10.1109/COMST.2015.2481183>.

DEBIAN. **Afinal de contas, o que é o Debian?** 2018. Disponível em: <https://www.debian.org/intro/about>.

DESHPANDE, U.; KEAHEY, K. Traffic-sensitive live migration of virtual machines. In: **2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing**. IEEE, 2015. Disponível em: <https://doi.org/10.1109/ccgrid.2015.163>.

DHANOA, I. S.; KHURMI, S. S. Analyzing energy consumption during VM live migration. In: **International Conference on Computing, Communication & Automation**. IEEE, 2015. Disponível em: <https://doi.org/10.1109%2Fccaa.2015.7148475>.

DOCKER. **What is Docker**. 2017. Disponível em: <https://www.docker.com/what-docker>.

DUARTE, O. C. M. B. et al. **Componentes Básicos do SNMP**. 2018. Disponível em: <https://www.gta.ufrj.br/grad/10_1/snmp/componentes.htm>.

EUGSTER, P. T. et al. The many faces of publish/subscribe. **ACM Computing Surveys**, Association for Computing Machinery (ACM), v. 35, n. 2, p. 114–131, jun 2003. Disponível em: <https://doi.org/10.1145/857076.857078>.

FORSTER, F. **Collectd – The system statistics collection daemon**. 2018. Disponível em: <https://collectd.org/>.

FORSTER, F. **Features Collectd**. 2018. Disponível em: <https://collectd.org/features.shtml>.

FORSTER, F. **Plugin: SNMP**. 2018. Disponível em: <https://collectd.org/wiki/index.php/Plugin:SNMP>.

GANDHI, A. et al. AutoScale. **ACM Transactions on Computer Systems**, Association for Computing Machinery (ACM), v. 30, n. 4, p. 1–26, nov 2012. Disponível em: <https://doi.org/10.1145/2382553.2382556>.

GANGADHARAN, G. Open source solutions for cloud computing. **Computer**, Institute of Electrical and Electronics Engineers (IEEE), v. 50, n. 1, p. 66–70, jan 2017. Disponível em: <https://doi.org/10.1109/mc.2017.20>.

GOOGLE. **Data Centers**. 2017. Disponível em: <https://www.google.com/about/datacenters/inside/locations/hamina/>.

GOOGLE. **Efficiency: How we do it**. 2017. Disponível em: <https://www.google.com/about/datacenters/efficiency/internal/index.html#water-and-cooling>.

HEIDARI, P.; LEMIEUX, Y.; SHAMI, A. Qos assurance with light virtualization - a survey. In: **2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)**. IEEE, 2016. Disponível em: <http://dx.doi.org/10.1109/CloudCom.2016.0097>.

HINES, M. R.; DESHPANDE, U.; GOPALAN, K. Post-copy live migration of virtual machines. **ACM SIGOPS Operating Systems Review**, Association for Computing Machinery (ACM), v. 43, n. 3, p. 14, jul 2009. Disponível em: <https://doi.org/10.1145/1618525.1618528>.

INTEL et al. Ipmi: Intelligent platform management interface specification second generation v2.0. p. 644, 2013. Disponível em: <http://www.intel.com.br/content/www/br/pt/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html>.

ITEAD. **Sonoff Pow**. 2018. Disponível em: <https://www.itead.cc/wiki/Sonoff_Pow>.

JOHANSSON, A.; AXELSSON, M.; GUSTAVSSON, K. Heuristic approach of exact bin-packing model. In: **2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)**. IEEE, 2017. Disponível em: <https://doi.org/10.1109/ieem.2017.8290051>.

KAUR, K. et al. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. **IEEE Wireless Communications**, Institute of Electrical and Electronics Engineers (IEEE), v. 24, n. 3, p. 48–56, 2017. Disponível em: <https://doi.org/10.1109/mwc.2017.1600427>.

KAVANAGH, R.; ARMSTRONG, D.; DJEMAME, K. Accuracy of energy model calibration with IPMI. In: **2016 IEEE 9th International Conference on Cloud Computing (CLOUD)**. IEEE, 2016. Disponível em: <https://doi.org/10.1109%2Fcloud.2016.0091>.

KOMINOS, C. G.; SEYVET, N.; VANDIKAS, K. Bare-metal, virtual machines and containers in openstack. In: **2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)**. IEEE, 2017. Disponível em: <http://dx.doi.org/10.1109/ICIN.2017.7899247>.

KOZAK, T.; PREDKI, P.; MAKOWSKI, D. Real-time IPMI protocol analyzer. **IEEE Transactions on Nuclear Science**, Institute of Electrical and Electronics Engineers (IEEE), v. 58, n. 4, p. 1857–1863, aug 2011. Disponível em: <https://doi.org/10.1109/tns.2011.2145000>.

LEANGSUKSUN, C. et al. Ipmi-based efficient notification framework for large scale cluster computing. In: **Sixth IEEE International Symposium on Cluster Computing and the Grid**. IEEE, 2006. Disponível em: <https://doi.org/10.1109/ccgrid.2006.1630918>.

LEHRIG, S.; EIKERLING, H.; BECKER, S. Scalability, elasticity, and efficiency in cloud computing. In: **Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures**. ACM Press, 2015. Disponível em: <https://doi.org/10.1145/2737182.2737185>.

MACHEN, A. et al. Live service migration in mobile edge clouds. **IEEE Wireless Communications**, Institute of Electrical and Electronics Engineers (IEEE), p. 2–9, 2017. Disponível em: <https://doi.org/10.1109/mwc.2017.1700011>.

MAGALHAES, D.; SOARES, J. M.; GOMES, D. Análise do impacto de migração de máquinas virtuais em ambiente computacional virtualizado. **XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, Único, p. 14, may 2011.

MANU et al. Docker container security via heuristics-based multilateral security-conceptual and pragmatic study. In: **2016 International Conference on Circuit Power and Computing Technologies (ICCPCT)**. IEEE, 2016. Disponível em: <http://dx.doi.org/10.1109/ICCPCT.2016.7530217>.

MASTELIC, T. et al. Cloud computing: survey on energy efficiency. **ACM Computing Surveys**, Association for Computing Machinery (ACM), v. 47, n. 2, p. 1–36, dec 2014. Disponível em: <https://doi.org/10.1145/2656204>.

MELHEM, S. B. et al. Markov prediction model for host load detection and VM placement in live migration. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 6, p. 7190–7205, 2018. Disponível em: <https://doi.org/10.1109/access.2017.2785280>.

MICROSOFT. **Project Natick**. 2014. Disponível em: <http://natick.research.microsoft.com>.

MIKE, H. **Request for Comments: 4181**. 2005. Disponível em: <https://tools.ietf.org/html/rfc4181>.

MORAES, R. F. de L. **O centro de dados do Facebook na Suécia que quase ninguém conhece**. 2014. Disponível em: <http://www.tecmundo.com.br/facebook/58364-centro-dados-facebook-suecia-ninguem-conhece.htm>.

MORRIS, J. et al. **Wall-e**. 2018. Disponível em: <https://www.pixar.com/feature-films/walle/#walle-1>.

MOSCATO, F. Exploiting semantics and patterns for verification of orchestrated cloud services. In: **2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems**. IEEE, 2015. Disponível em: <https://doi.org/10.1109/cisis.2015.26>.

NEJAD, E. S. et al. Infrastructure of data centers for transferring big data traffic: A survey research. In: **2015 International Congress on Technology, Communication and Knowledge (ICTCK)**. Institute of Electrical and Electronics Engineers (IEEE), 2015. Disponível em: <http://dx.doi.org/10.1109/ICTCK.2015.7582717>.

NIELSEN, J. **Response Times: The 3 Important Limits**. 1993. Disponível em: <https://www.nngroup.com/articles/response-times-3-important-limits/>.

OPENSTACK. **Ironic**. 2017. Disponível em: <https://wiki.openstack.org/wiki/Ironic#OpenStack_Bare_Metal_Provisioning_Program>.

OPENSTACK. **OpenStack Telemetry**. 2017. Disponível em: <https://wiki.openstack.org/wiki/Telemetry>.

OPENSTACK. **Overview**. 2017. Disponível em: <https://docs.openstack.org/ocata/install-guide-ubuntu/overview.html>.

OPENSTACK. **Software: What is OpenStack?** 2017. Disponível em: <https://www.openstack.org/software/>.

OPENSTACK. **Welcome to Ceilometer documentation**. 2017. Disponível em: <https://docs.openstack.org/ceilometer/latest/>.

OPENSTACK. **Configure live migrations**. 2018. Disponível em: <https://docs.openstack.org/nova/pike/admin/configuring-migrations.html>.

OPENSTACK. **Glossary**. 2018. Disponível em: <https://docs.openstack.org/ocata/networking-guide/common/glossary.html>.

OPENSTACK. **OpenStack Installation Tutorial for Ubuntu**. 2018. Disponível em: <https://docs.openstack.org/ocata/install-guide-ubuntu/index.html>.

OPENSTACK. **Configure live migrations**. 2019. Disponível em: <https://docs.openstack.org/ocata/admin-guide/compute-configuring-migrations.html>.

ORACLE. **Reference Manual: What is MySQL?** 2017. Disponível em: <https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>.

ORGERIE, A.-C.; LEFEVRE, L.; GELAS, J.-P. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In: **2008 14th IEEE International Conference on Parallel and Distributed Systems**. IEEE, 2008. Disponível em: <https://doi.org/10.1109/icpads.2008.97>.

PEREZ, X. **ESPurna Firmware**. 2018. Disponível em: <https://github.com/xoseperez/espurna>.

PHPGROUP. **Prefácio**. 2017. Disponível em: <http://php.net/manual/pt_BR/preface.php>.

PIRAGHAJ, S. F. et al. A framework and algorithm for energy efficient container consolidation in cloud data centers. In: **2015 IEEE International Conference on Data Science and Data Intensive Systems**. IEEE, 2015. Disponível em: <https://doi.org/10.1109/dsdis.2015.67>.

REDHAT. **O que é um container Linux?** 2019. Disponível em: <https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container>.

ROSSIGNEUX, F. et al. A generic and extensible framework for monitoring energy consumption of OpenStack clouds. In: **2014 IEEE Fourth International Conference on Big Data and Cloud Computing**. Institute of Electrical & Electronics Engineers (IEEE), 2014. Disponível em: <http://dx.doi.org/10.1109/BDCloud.2014.105>.

SAHASRABUDHE, S. S.; SONAWANI, S. S. Comparing openstack and VMware. In: **2014 International Conference on Advances in Electronics Computers and Communications**. Institute of Electrical and Electronics Engineers (IEEE), 2014. Disponível em: <http://dx.doi.org/10.1109/ICAECC.2014.7002392>.

SAHNI, S.; VARMA, V. A hybrid approach to live migration of virtual machines. In: **2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)**. IEEE, 2012. Disponível em: <https://doi.org/10.1109/ccem.2012.6354587>.

SENSU. **Sensu Architecture**. 2018. Disponível em: <https://docs.sensu.io/sensu-core/1.2/overview/architecture/#sensu-architecture>.

SENSU. **Sensu: Full-stack monitoring for today's business**. 2018. Disponível em: <https://sensuapp.org/>.

SENSU. **Sensu: Plugins SNMP**. 2018. Disponível em: <https://github.com/sensu-plugins/sensu-plugins-snmp>.

SHAH, S. A. R.; JAIKAR, A. H.; NOH, S.-Y. A performance analysis of precopy, postcopy and hybrid live VM migration algorithms in scientific cloud computing environment. In: **2015 International Conference on High Performance Computing & Simulation (HPCS)**. IEEE, 2015. Disponível em: <https://doi.org/10.1109/hpcsim.2015.7237044>.

SHAO, C. et al. OpenStack platform and its application in big data processing. In: **2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)**. Institute of Electrical and Electronics Engineers (IEEE), 2015. Disponível em: <http://dx.doi.org/10.1109/ICINIS.2015.45>.

SHARKH, M. A. et al. Simulating high availability scenarios in cloud data centers: A closer look. In: **2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)**. IEEE, 2015. Disponível em: <https://doi.org/10.1109/cloudcom.2015.62>.

SINGH, R. et al. Yank: Enabling green data centers to pull the plug. In: **Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)**. Lombard, IL: USENIX, 2013. p. 143–155. ISBN 978-1-931971-00-3. Disponível em: <https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/singh>.

SONG, T.; YOSHIHIRO, K.; ASAMITOHRU. Cache management algorithm of load balancer for large-scale snmp monitoring system. In: **2013 IEEE Globecom Workshops (GC Wkshps)**. IEEE, 2013. Disponível em: <http://dx.doi.org/10.1109/GLOCOMW.2013.6825104>.

SPITZCOVSKY, D. **Facebook construirá data center próximo ao Ártico para economizar energia**. 2011. Disponível em: <http://super.abril.com.br/blog/planeta/facebook-construira-data-center-proximo-ao-artico-para-economizar-energia/>.

STENBERG, D.; FANDRICH, D. **Command line tool and library for ransferring data with URLs**. 2018. Disponível em: <https://curl.haxx.se/>.

TAY, Y.; GAURAV, K.; KARKUN, P. A performance comparison of containers and virtual machines in workload migration context. In: **2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)**. IEEE, 2017. Disponível em: <https://doi.org/10.1109/icdcsw.2017.44>.

TONG, G. et al. Measuring and analyzing CPU overhead of virtualization system. In: **2011 IEEE Asia-Pacific Services Computing Conference**. IEEE, 2011. Disponível em: <https://doi.org/10.1109/apscc.2011.40>.

UNION, E. **Data Centre Industry**. 2017. Disponível em: <http://www.geyser-project.eu/data-centre-industry>.

UPADHYAY, A.; LAKKADWALA, P. Secure live migration of vms in cloud computing: A survey. In: **Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization**. IEEE, 2014. Disponível em: <https://doi.org/10.1109/icrito.2014.7014766>.

VALLEE, G. et al. System-level virtualization for high performance computing. In: **16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)**. Institute of Electrical and Electronics Engineers (IEEE), 2008. Disponível em: <https://doi.org/10.1109%2Fpdp.2008.85>.

VARASTEH, A.; GOUDARZI, M. Server consolidation techniques in virtualized data centers: A survey. **IEEE Systems Journal**, Institute of Electrical and Electronics Engineers (IEEE), p. 1–12, 2015. Disponível em: <https://doi.org/10.1109%2Fjsyst.2015.2458273>.

VERAS, M.; KASSICK, R. Virtualização de servidores. p. 422, jan 2011. RNP/ESR.

WADHWA, B.; VERMA, A. Energy and carbon efficient VM placement and migration technique for green cloud datacenters. In: **2014 Seventh International Conference on Contemporary Computing (IC3)**. Institute of Electrical and Electronics Engineers (IEEE), 2014. Disponível em: <http://dx.doi.org/10.1109/IC3.2014.6897171>.

WAN, T.-Y. et al. Implementation of an energy saving cloud infrastructure with virtual machine power usage monitoring and live migration on OpenStack. In: **2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)**. IEEE, 2016. Disponível em: <https://doi.org/10.1109/icpads.2016.0101>.